Theses and Dissertations   1. Thesis and Dissertation Collection, all items

1990-06

Designing intelligent computer aided instruction systems with integrated knowledge representation schemes

Ragsdale, Daniel Joseph; Tidd, John Patrick

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/37532

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AD-A224 178

DESIGNING INTELLIGENT COMPUTER AIDED INSTRUCTION SYSTEMS WITH INTEGRATED KNOWLEDGE REPRESENTATION SCHEMES

by

Daniel J. Ragsdale
and
John P. Tidd

June 1990

Thesis Advisor: Yuh-jeng Lee

Approved for public release; distribution is unlimited.

90 07 16 332

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School

6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code)

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

| 8c. ADDRESS (City, State, and ZIP Code) | 10. SOURCE OF FUNDING NUMBERS |

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DESIGNING INTELLIGENT COMPUTER AIDED INSTRUCTION SYSTEMS WITH INTEGRATED KNOWLEDGE REPRESENTATION SCHEMES

12. PERSONAL AUTHOR(S)
Tidd, John P. and Ragsdale, Daniel J.

13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) June 1990	15. PAGE COUNT 111

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Intelligent Computer Aided Instruction, Knowledge Representation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
Over the past twenty years, automated tutoring systems have gained an increasing recognition as a prominent area of Artificial Intelligence (AI). During this period, Intelligent Computer Aided Instruction (ICAI) systems have been developed using a variety of AI techniques to enhance the learning process. The core AI issue in designing these systems concerns knowledge representation. A review of current AI literature shows that there are numerous, distinctly different knowledge representation schemes, and most conventional programming environments do not readily support all of these representation schemes.

This thesis proposes that tutoring systems are best designed in a programming environment that supports multiple, integrated knowledge representation schemes. Such an environment allows the designer to select and employ, with ease, the most natural knowledge representation scheme for each type of knowledge in the tutoring system. In this thesis we describe the components of a generalized ICAI system; discuss the various types of knowledge and knowledge representation schemes; and review the knowledge representation schemes used in several noted ICAI systems. In addition, we describe two prototype ICAI systems (Map

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Yuh-jeng Lee	22b. TELEPHONE (Include Area Code) (408) 646-2361 · 22c. OFFICE SYMBOL 52LE

Reading Tutor and Pilot Emergency Procedure Tutor) which we designed and developed in a specific programming environment that supports multiple, integrated knowledge representation schemes. We also analyzed the behavior of both systems and discussed how the programming environment has facilitated the construction process of the two prototypes and enhanced their overall quality.

DESIGNING INTELLIGENT COMPUTER AIDED INSTRUCTION SYSTEMS WITH INTEGRATED KNOWLEDGE REPRESENTATION SCHEMES

by

Daniel Joseph Ragsdale
Captain, United States Army
B.S., United States Military Academy, 1981

and

John Patrick Tidd
Captain, United States Army
B.S., United States Military Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Authors: _____
Daniel Joseph Ragsdale

_____
John Patrick Tidd

Approved By: _____
Yuh-jeng Lee, Thesis Adviser

_____
MAJ George E. Thurmond, Second Reader

_____
Robert B. McGhee, Chairman,
Department of Computer Science

Accession For	
NTIS GRA&I	X
DTIC TAB	□
Unannounced	□
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

ii

ABSTRACT

Over the past twenty years, automated tutoring systems have gained an increasing recognition as a prominent area of Artificial Intelligence (AI). During this period, Intelligent Computer Aided Instruction (ICAI) systems have been developed using a variety of AI techniques to enhance the learning process. The core AI issue in designing these systems concerns knowledge representation. A review of current AI literature shows that there are numerous, distinctly different knowledge representation schemes, and most conventional programming environments do not readily support all of these representation schemes.

This thesis proposes that tutoring systems are best designed in a programming environment that supports multiple, integrated knowledge representation schemes. Such an environment allows the designer to select and employ, with ease, the most natural knowledge representation scheme for each type of knowledge in the tutoring system. In this thesis we describe the components of a generalized ICAI system; discuss the various types of knowledge and knowledge representation schemes; and review the knowledge representation schemes used in several noted ICAI systems. In addition, we describe two prototype ICAI systems (Map Reading Tutor and Pilot Emergency Procedure Tutor) which we designed and developed in a specific programming environment that supports multiple, integrated knowledge representation schemes. We also analyzed the behavior of both systems and discussed how the programming environment has facilitated the construction process of the two prototypes and enhanced their overall quality.

iv

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I. INTRODUCTION

A. BACKGROUND

Throughout the 20th century educators have been intrigued with the possibility of employing automated tools in the classroom. By the 1950's a complete industry developed in this country which provided schools and industry with surprisingly sophisticated mechanical devices, referred to as "teaching machines", to assist educators in teaching/training. The advent of the commercially available digital computers in the late 1950's resulted in a new type of automated teaching aid, Computer Aided Instruction (CAI) systems, which soon became a common application for those early computer systems. At the time many educators and computer scientists believed that CAI was a panacea for many of the problems that plagued our school systems. Consequently, they made exaggerated claims about the short term potential for CAI systems to resolve many of these perceived problems. The pioneers of CAI went as far as claiming that "through this medium highly individualized (and effective) instruction would soon be common-place" (Sleeman, 1984, p. 2). Unfortunately, most of the early CAI systems failed to live up to the high expectations of their proponents and most were considered miserable failures by their users. Probably the best description of these early systems is that they were nothing more then sophisticated, electronic "page turners" (Barr and Feigenbaum, 1982 , p. 225).

By the late 1960's it was clear to most observers of the field that a new approach was needed if the CAI field was to realize its potential. By 1970 many CAI researchers gained better appreciation for the great complexity associated with the process of providing effective, individualized instruction. A number of these research-

1

ers, particularly those with a background in computer science, concluded that because of the extreme complexity and the ill-defined nature of the instructional process that artificial intelligence techniques should be employed in the design and development of such systems. This conclusion led to research which resulted in an entirely new type of tutoring systems, dubbed Intelligent Computer Aided Instruction (ICAI). Jaime Carbonell produced the first ICAI system in 1970 when he created his "mixed-initiative" geography tutor, SCHOLAR (Carbonell, 1970). Since Carbonell's seminal work many, widely varying ICAI systems have been developed.

Most of these intelligent systems took several years to develop but, unfortunately very few have left the laboratory. Despite this lack of commercial success, much has been learned through the development of these ICAI systems. One of the most important lessons to come out of this work was the realization that, in order to be effective, ICAI systems must "capture" and represent knowledge about a number of distinctly different domains. Instructional system designers also learned that they needed to represent knowledge concerning not only the domain in which they were providing instruction, but also knowledge about teaching (pedagogical issues), and about the student (cognitive issues) (Pliske and Psotka, 1986, pp. 52-53). In addition, another type of knowledge, knowledge about communication (interface issues) must also be represented in the system.

Since the knowledge contained in these various domains is very complex and frequently ill-defined, the early ICAI researchers discovered that they needed to represent distinctly different forms of knowledge within their systems. For example, within a typical ICAI system the following forms of knowledge must be represented in some manner: declarative knowledge, procedural knowledge, and meta-cognitive knowledge (Dede, 1986, p. 333). Because of the diversity of knowledge domains and

forms of knowledge, the selection of the knowledge representation method(s) is clearly a significant issue in the design of ICAI systems.

The development of alternative methods of representing knowledge has always been a central issue for AI researchers. Because of this great emphasis, in the past 35 years AI researchers have developed many different knowledge representation methodologies to support their computational requirements. These methodologies or schemes include: formal logic-based representations, procedural representations, associative representations, production systems, structured object representations, and other special purpose representation schemes (Barr & Feigenbaum, 1982, pp. xiii - xix). Each representation scheme has its strengths and weaknesses, and is usually well suited for representing only a subset of the knowledge types. Consequently, since an effective ICAI system must capture various types of knowledge from a number of different domains, the programming language/environment in which these systems are designed should support multiple knowledge representation schemes. In addition these multiple schemes should be fully integrated to allow the designer to work with many different forms of knowledge simultaneously. Unfortunately, while it is theoretically feasible to implement all of the knowledge schemes mentioned above in any programming language/environment, there are actually only a few programming environments which readily support multiple, integrated, knowledge representation schemes. Most AI researchers believe that the selection of the most natural representation schemes for the knowledge in a given application is one of the most important design decision made by an AI system developer. Consequently, the process of designing and developing an effective ICAI system is unnecessarily impeded if it is done in a programming environment which does not support multiple, integrated knowledge representation schemes.

B. OBJECTIVE

The purpose and intent of this thesis is to demonstrate that the most suitable programming environment for designing and developing an Intelligent Computer Aided Instruction system is one which supports multiple, integrated knowledge representation schemes. In order to support this claim we have designed and implemented two separate ICAI system prototypes in such an environment.

C. ORGANIZATION

Chapters II and III of this thesis provide overviews of generalized ICAI systems and knowledge representation schemes, respectively. Chapter IV discusses knowledge representation issues as they relate to specific ICAI systems. The following chapter presents an overview of the development environment we used to design and construct two ICAI applications. Chapters VI and VII describe the ICAI systems that we developed in detail. Finally, Chapter VIII discusses how the availability of multiple integrated knowledge representation schemes enhanced the development of both applications. The appendices contain sample tutoring sessions of the two applications and a description of where interested readers may gain access to the source code.

II. INTELLIGENT COMPUTER AIDED INSTRUCTION SYSTEMS

A. INTRODUCTION

The history of Computer Aided Instruction began in the early 1960's and during the past thirty years the technology has matured and the demand for such systems has increased greatly. In response to this demand, a number of powerful CAI authoring tools/environments were developed. The most notable of these tools include PLATO, TICCIT, and WICAT (Kearsley, 1987, p. 29). Employment of these and other authoring tools resulted in a significant reduction in the overall difficultly of designing and implementing CAI systems. Because of this reduction in difficulty and continued demand, over ten thousand CAI systems, generally referred to as "courseware" have been developed (Anderson et al, 1985, p. 456). To the credit of the designers of these systems, some systems provide adequate instruction in very restricted domains. Unfortunately, the overwhelming majority of these systems suffer from severe limitations.

B. FROM CAI TO ICAI

The primary motivation for the researchers who developed the first ICAI systems were the numerous shortcomings which were evident in existing CAI systems. These shortcomings have been described by many researchers working in the ICAI field (Sleeman, 1984; Pliske, 1986; Carbonell, 1970). Marlene Jones provided an especially detailed enumeration of these limitations (Jones, 1984, p. 517). She claims that most CAI systems suffer form the following limitations:

(i)   an inability to conduct conversations with the student in the students natural language;

(ii)   an inability to understand the subject matter being taught, thus being able to anticipate responses (or answer student questions);

(iii) an inability to decide (reason about) what should be taught next;

(iv) an inability to anticipate, diagnose, and understand the student's mistakes and misconceptions;

(v)   an inability to improve or modify current teaching strategies or learn new ones.

These shortcomings and others were clearly obvious to nearly everyone who worked in the field of computer aided instruction. Unfortunately, the majority of researchers working in the CAI field were not aware of, or did not know how to employ, the "tools" that are necessary to address/mitigate these shortcomings.

On the other hand, AI researchers, particularly those working in the areas of natural language understanding, expert systems, machine learning, and knowledge representation, were able to recognize the clear correspondence between their research interests and the limitations of conventional CAI systems. A number of these researchers applied various AI techniques to the development of CAI systems in attempts to alleviate some of the acknowledged shortcomings. The goal of these researchers was to improve the quality and effectiveness of computer aided instruction systems by designing systems which more closely simulated the instructional process that takes place between a human tutor and student (i.e., systems which acted in a more intelligent manner). The result of their combined efforts was a new class of instructional sytem; appropriately named Intelligent Computer Aided Instructional systems.

C. COMPARISON BETWEEN CAI SYSTEMS AND ICAI SYSTEMS

Although ICAI systems were first developed to improve upon and extend the capabilities of "conventional" CAI systems, there are a number of fundamental differences between these two types of instructional systems. A description of the differences

6

between the two system types is presented here to provide the reader with a better overall understanding of ICAI systems. Greg Kearsley provides an excellent description of these differences (Kearsley, 1987, pp. 3-10). He argues that typical ICAI and CAI systems differ in the following areas: development goals, theoretical bases of the designers, instructional principles, methods of structuring knowledge (representation), methods of student modeling, subject matter areas, system development process, and finally, hardware and software differences. Most of these differences can be attributed to the contrasting backgrounds of the respective developers of these two types of systems. Conventional CAI systems are normally designed by educational and psychology researchers. ICAI systems, on the other hand are usually developed exclusively by AI researchers. The designers of ICAI systems are typically most interested in do empirical testing to determining the desirability of applying a particular AI technique to enhance the learning process. Conversely, CAI developers are primarily interested in developing marketable, commercial tutoring systems. In other words, ICAI developers are mostly concerned with the "means" while CAI developers are primarily concerned with the "ends." (Kearsley, 1987, p. 3-10)

It should be noted that in recent years CAI developers have begun to incorporate many of the lessons learned in the development of ICAI system. Consequently, the line which divides the two types of systems is becoming somewhat blurred (Wegner, 1987, p. 5). In the absence of any other indicators, a determination of whether or not an instructional system is an ICAI can be based on two important factors. First, if the system is both a dynamically adaptive learning environment, and second, if the system appears to "know" (i.e., can reason about) the knowledge it is attempting to teach, then it can be considered an ICAI. If, on the other hand, the system appears to be presenting a pre-programed sequence of text and graphics and can not dynamically

adapt to meet the instructional needs of individual students then the system can not be considered an ICAI.

Many authors use a more strict definition of ICAI systems which includes the requirement that the systems adhere to a particular architecture (i.e., that it must have a particular set of components) before it can be considered an ICAI. We don't believe that such a restriction should be placed on the definition; however, the architectures that these authors propose provides an excellent framework for describing the components found in most existing ICAI systems.

D. COMPONENTS OF ICAI SYSTEMS

Providing adaptive and individualized instruction is an extremely complex process which involves two agents; a student and a tutor, a body of knowledge, and a communication medium. Because of this obvious complexity, the first group of AI researchers who worked in the field concluded that it should be broken down into logical modules. As early as 1973 Hartley and Sleeman concluded that a generalized ICAI should have at least four distinct components: (1) knowledge of the domain, (2) a model of the student, (3) a set of teaching operations, and (4) a set of teaching guidance rules (Park, 1987, p. 274). In recent years many other authors have proposed a variety of models to describe generalized ICAI systems (Seidel, 1988; Woolf, 1987; Tennyson, 1987; Nawrocki, 1987). These various models appear to have seemingly very different component schemes, however, most of the models differ only in a cosmetic manner.

The most widely accepted model of a generalized ICAI consists of the following modules/components: the domain knowledge model, the student model, the tutor model, and the communication model (Woolf, 1987, pp. 1-44). Note that the only difference between this model and the original model proposed by Hartley is that the third

8

and fourth components of Hartley's model have been combined into one component and a new component, the communications model, has been added. Figure 1 shows the model and the interaction among the various components. It should be noted that in most ICAI systems one or more of these modules are either not fully developed or nonexistent. The reason for this "shortcoming" is that most ICAI researchers implemented their systems in order to validate a specific hypotheses and consequently they usually focused on one particular module, at the expense of the others (Duchastel, 1986, p.103).

```
┌─────────────────────────────────────────────────────────────┐
│            ┌─────────────────────────┐                       │
│            │   DOMAIN KNOWLEDGE      │                       │
│            │        MODEL            │                       │
│            └─────────────────────────┘                       │
│                                                               │
│   ┌──────────────┐          ┌──────────────┐    ┌─────────┐  │
│   │ TUTOR MODEL  │ ───────▶ │ COMMUNICATION│◀── │ STUDENT │  │
│   │              │          │    MODEL     │──▶ │         │  │
│   └──────────────┘          └──────────────┘    └─────────┘  │
│                                                               │
│            ┌─────────────────────────┐                       │
│            │     STUDENT MODEL       │                       │
│            └─────────────────────────┘                       │
└─────────────────────────────────────────────────────────────┘
```
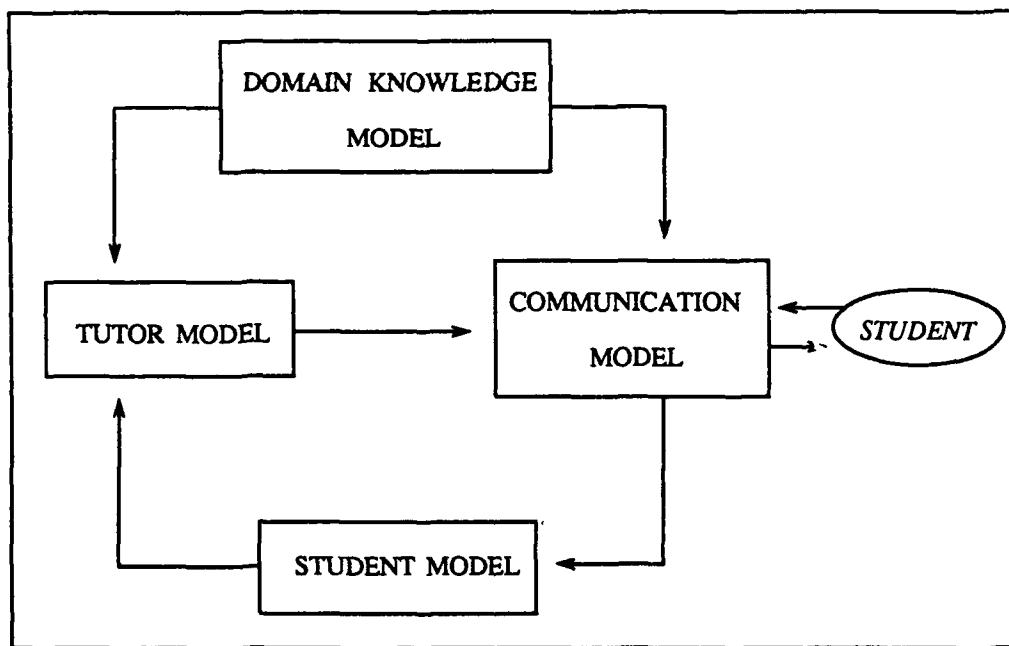
Figure 1 - Model of a Generalized ICAI System

1. Domain Knowledge Model

The domain knowledge model is the component of the system in which the subject matter expertise is represented. All tutoring systems, including both CAI and ICAI systems, have some form of knowledge model. The knowledge model of a typi-

cal ICAI is significantly more sophisticated than the corresponding module in a CAI. In most CAI systems, for example, the knowledge model is merely a description of concepts and skills that a student should acquire. This coarsely grained and extremely restricted form of knowledge representation prevents CAI systems from reasoning with/about the domain knowledge. The knowledge models of most ICAI systems, on the other hand, explicitly supports reasoning and inferences about the domain knowledge by the system. In addition, many ICAI systems represent the domain knowledge as a sophisticated, executable model thereby making a "dynamic" form of the knowledge available to the system (Wegner, 1987, p. 14).

In an ICAI system, the domain knowledge model serves three distinct purposes. First, the domain knowledge model is the source of the knowledge which is being imparted to the student. Second, the domain knowledge model provides a solution to the problem that the system has posed to the student. Third, the domain knowledge model reasons about and prepares responses to questions posed by the student (note: this requirement does not imply that all ICAI must have a natural language interface. In some reactive environments the student poses a "question" by taking some action which alters the state of the system). AI researchers have employed a number of AI techniques to provide this functionality to their domain knowledge models. (Wegner, 1987, pp. 14-15)

2. Student Model

The student model is the component of the system in which evidence concerning the knowledge that the student has acquired is gathered and represented. The student model can be either, a simple declarative record of the student's knowledge, or a sophisticated diagnostic model which evaluates the student's responses and infers the current state of the student's knowledge. In addition, the representation of a student's knowledge can be either quantitatively based or based on some form of

10

qualitative representation. In many ICAI systems, the student model represents the student's knowledge only as a subset of the expert's knowledge. In these systems the student model is referred to as an "overlay model." In other systems, the student model also represent student misconceptions about the knowledge domain. These models are called "buggy" models. (Kearsley, 1988, p. 17)

An extremely important consideration in the student model is the assignment of "credit" for correct responses and "blame" for incorrect responses. Since it is very difficult for human tutors to make this type of determination, this is an especially difficult problem for an automated system. Another issue which adversely affects the development of suitable student models is the fact that students do not always make mistakes because of a lack of knowledge. They also make mistakes for a variety of other reasons including carelessness and fatigue. Consequently many ICAI systems employ sophisticated techniques to handle this type of "noisy" data. (Sleeman and Brown, 1982, p. 4)

3. Tutor Model

The tutor model is the component of the system in which the pedagogical knowledge of the system is represented. The tutoring strategy that an ICAI system employs is typically encapsulated in the tutor model. The tutor model performs two distinct functions: (1) determination of the student's instructional needs, and (2) selection and presentation of domain knowledge to the student (Park, 1988, p. 9). In ICAI systems the tutoring model performs these functions by reasoning about the student's learning needs and about what to present to the student next. In most systems this reasoning is accomplished by interacting with the student model and through the execution of an explicit procedure or through the application of production rules.

11

Over the years ICAI developers have implemented a wide variety of tutoring strategies/approaches. Most of these approaches fall into two general categories: the Socratic method, and the coaching method (Kearsley, 1987, p. 17). In the Socratic method, the student is provided with questions which "guide" him through resolution of his misconceptions. In the coaching method, the student is provided with a reactive learning environment in which the coach interrupts only when necessary to insure that the student is learning new concepts as a "side-effect" of the students actions. Another common tutoring strategy, which does not fit into the two categories listed above, is the "discovery" learning method. In this method there is no explicit tutor model. Instead the student has full initiative and is not interrupted by the system at any time.

4. Communications Model

The communications model is the component of the system in which knowledge about the interface to the student is represented. The communications model serves two important functions: (1) interpretation of the students responses, and (2) display of the system's output to the screen (Park, 1987, p. 237). Many early ICAI researchers down played the importance of this module. In fact most of the early ICAI component models did not include an explicit communication model. The communication model has achieved greater relative importance over the years, due to the development of sophisticated and expressive graphical interfaces and improved natural language understanding systems.

In some systems natural language understanding and generation systems are employed. These systems control the discourse between the student and the system. Other systems have no natural language understanding system. These systems, instead, depend upon a sophisticated graphical interface to communicate with the student. Many current system employ a combination of these two techniques.

12

E. SUMMARY

There are some significant differences between conventional CAI and ICAI systems. In recent years, however, the line which divides the two types of systems is becoming blurred as more CAI system designers "borrow" AI techniques in the design and implementation of their systems.

The four component model of a generalized ICAI is widely accepted by contemporary ICAI researchers. Although there is considerable divergence of opinion about the particular functions that each module should perform, there is little dispute about the overall framework of the model. An understanding of the model and its four components provides a framework for discussing the knowledge representation issues which must be addressed in the design and implementation of an ICAI system.

III. KNOWLEDGE REPRESENTATION METHODOLOGIES

A. INTRODUCTION

Knowledge can be considered a collection of related facts, procedures, models and heuristics that can be used in problem solving or inference systems (Tanimoto, 1987 p. 80).

Knowledge and knowledge representation have always been fundamental issues in AI research. Many AI researchers, in fact, believe that knowledge representation is the single most important issue in AI (Barr & Feigenbaum, 1982, p. 159). One of the primary reasons for this belief is the fact that knowledge representation is a central concern in every sub-field of AI. During the early years of AI research, researchers realized that if they wished to design systems which exhibited intelligent behavior they needed to develop knowledge representation schemes which allowed their systems to "reason" about, and make inferences from, the knowledge that the system uses.

It should be noted that AI researchers did not "invent" the knowledge representation issue, and the knowledge representation issue not a new concern. In fact, mankind has been wrestling with this issue since the time of Aristotle (Brachman & Levesque, 1985, p. xiv). Further, the theories upon which many knowledge representation schemes are based were developed many years prior to the invention of digital computers. For example, the logic theories of Boole, Frege, and Russel, which were devised in the eighteenth century, are the theoretical foundations of all logic representation schemes (Barr & Feigenbaum, 1982, p. 160).

B. TYPES OF KNOWLEDGE

It is widely accepted that knowledge can take on many different forms and that these different forms can be categorized in a number of different ways. Unfortunately there is little agreement among AI researchers as to the manner in which knowledge can be categorized. For example, four different forms of knowledge are described in the Handbook of Artificial Intelligence (Barr and Feigenbaum, 1982, p. 146): knowledge about objects, knowledge about events, knowledge about performance and knowledge about knowledge. On the other hand, Beverley Woolf, an authority in the ICAI field, believes there are three types of knowledge: conceptual knowledge, procedural knowledge and heuristic knowledge (Woolf, 1986, p. 47). A number of other authors have proposed different category breakdowns for knowledge (Gevarter, 1978; Dede, 1986).

We will use a classification system for the types of knowledge which is based upon the categorizations mentioned above. In this classification system, all knowledge is divided into the following general categories: declarative knowledge, procedural knowledge and meta-knowledge. Later we will use this categorization as the basis for our discussion of knowledge representation schemes.

Declarative knowledge is factual information about physical objects and abstract concepts. Specifically, it defines the static state of objects or concepts. In simple terms, declarative knowledge can be used to answer the "what" question. The name height and weight of a person are examples of declarative knowledge.

Procedural knowledge is the information concerning actions and events in the physical world. The actions that procedural knowledge describes can be considered to be operations which cause changes or transitions to the static state of objects. In other words, procedural knowledge is used primarily to answers the "how" question. An example of procedural knowledge is a description of the process of driving a car.

15

Meta knowledge is knowledge about, or concerning, knowledge. For the purposes of definition the following types of knowledge (and others) can be classified as meta-knowledge: heuristics, certainties, and constraints. Heuristics, for example, can be considered to be knowledge about when to apply certain procedures. Certainties and constraints, on the other hand, can be considered to be knowledge about declarative facts.

C. KNOWLEDGE REPRESENTATION METHODOLOGIES

Through the years, AI researchers have developed a number of distinctly different knowledge representation schemes. In this section, we present five of the more prominent and historically significant categories of representation schemes. They are: formal logic-based representations, associative representations, procedural representations, production systems, and structured object representations. Most of the representation schemes that have been proposed by AI researchers during the past thirty years fall into these five general categories (Brachman and Levesque, 1985).

It should be noted that many of the proponents of each category of representation schemes believe that their scheme should be used exclusively for representing all knowledge. The well known and hard fought "proceduralist vs. declarativist" debate was based upon such extreme beliefs (Winograd, 1975, p. 358). In this thesis, however, we take the less extreme position that some types of knowledge are best represented by a particular representation scheme, while other types of knowledge might be better represented by another representation scheme. It is also our belief that some knowledge can be best represented by multiple representation schemes. In addition, we believe that the manner in which the knowledge is used by the system plays a large role in determining the most suitable representation scheme for a specific body of knowledge.

16

A survey of the five general categories of representation methodologies follows this section. A general description of each methodology and the strengths and weaknesses of each will be presented. The different representation schemes will be described and compared against each other using the following set of characteristics of representation schemes: modularity, understandability, completeness, consistency, grain size, and flexibility. (Winograd, 1975, p. 359; Barr & Feigenbaum, 1982, p. 147)

1. Formal Logic-based Representations

Formal logic has long been used to represent knowledge. One of the founding fathers of AI, John McCarthy, was an early proponent for the development of a logical formalism for representing knowledge. Using propositional logic, the simplest form of logic, all knowledge is represented as a set of declarative facts. These declarative facts are referred as axioms and are normally in the form of statements or logical sentences which adhere to a strict syntax. An inference mechanism is applied to these axioms to determine if other statements can be proved or inferred. This inference mechanism is based on the formal rules of logic. In predicate logic-based representation schemes, the expressiveness of the formalism is extended to include both predicates and the additional inference mechanism necessary to operate on these more complex statements. First-order logic systems have been extended further to include the concepts of functions and equality of predicates (Barr & Feigenbaum, 1982, pp. 160-171; McCarthy, 1977, p. 24).

There are many advantages using logical representation for knowledge. First, it is complete and sound, i.e., all true statements can be proved and all proved statements are true. Another important advantage of this scheme is the modularity and fine granularity of the knowledge that can be represented using formal logic. This modularity and fine granularity results from the fact that each new statement that is added into the system is completely independent of any other axiom. Still another advan-

tage of logic-based representation schemes is the flexibility that this scheme provides in the manner in which facts are used. (Barr & Feigenbaum, 1982, p. 160-171)

Despite these advantages, formal logic-based systems suffer from a number of serious disadvantages. First, the introduction of a large number of logical sentences into such a system often results in a "combinatorial explosion", due to lack of structure in a logic representation. Efforts to restrict the reasoning process has only mitigated, but not solved this problem. Another serious disadvantage of this scheme is that it is very difficult to group related facts or represent procedural knowledge.

There is yet another serious disadvantage associated with it the use of logic as the basis for a knowledge representation scheme. Although the reference mechanism is truth preserving, the semantic relationships between objects may be lost in the representation, resulting in logically valid, but semantically meaningless conclusions.

2. Associational Representations

M. Ross Quillian is credited with originating the concept of the semantic network upon which nearly all associational representation schemes are based. In Quillian's seminal dissertation he developed a representation scheme which was based upon his theoretical model of human long-term memory. According to his theory, the human memory consists of "a mass of nodes, interconnected by different kinds of associative links" (Quillian, 1967, p. 99). The nodes in his model usually represent concepts and the links represent specific relationships between two nodes in the model (Barr & Feigenbaum, 1982, p. 181).

Quillian developed his memory model, now commonly referred to as a semantic net, as the representation scheme for the knowledge in a natural language understanding system. Since he published his work in the late 1960's AI researchers have used semantic nets in a wide variety of applications. These applications include: com-

18

puter based psychological models, natural language understanding systems, data base management systems, and intelligent tutoring systems. Many of the researchers who used Quillian's theory have extended and modified the concept and, consequently, there are now numerous types of semantic networks. (Barr & Feigenbaum, 1982, p. 180)

Associative representation schemes offer a number of advantages. The principle advantage of this scheme is that it is easy to represent both the descriptive facts about a particular object or concept and its relationships with other objects or concepts. In addition, since a significant amount of human knowledge concerns the relationships that exist between objects or concepts this scheme readily supports the representation of this type of knowledge. Yet another advantage of this scheme is that the knowledge represented in a semantic net is usually in a readily understandable form. (Quillian, 1967, pp. 98-117)

Unfortunately, some problems result from the use of associational representation schemes. Since there are no formal inference rules, as there are in logic-based schemes, the conclusions inferred from the network are not guaranteed to be complete or consistent. In addition, it is very difficult to represent procedural knowledge in a semantic net. Further, associational representation schemes normally suffer from combinatorial problems as size and complexity of the system grows. (Barr & Feigenbaum, 1982, pp. 180-189)

3. Procedural Representations

Advocates of procedural representation schemes were motivated in their research by the inherent disadvantages associated with declarative-based representation schemes. Despite the claims of "declarativists", much of the knowledge that humans possess, and which AI researchers wish to represent in their systems, is procedural in nature. Terry Winograd made this point effectively when he stated "It is

19

an obvious fact that many things we know are best seen as procedures, and it is difficult to describe them in a purely declarative way" (Winograd, 1975, p. 187). Another important issue is the fact that all computer applications, including ICAI's, have at least some part of their knowledge represented in a procedural form.

The principle advantage to procedural representation schemes is very obvious: such schemes explicitly support the representation of procedural knowledge. In addition, because procedures are "direct in their problem solving activity", the combinatorial problems associated with logic-based and associational representations schemes are avoided. In addition, because of this "direct" nature procedural representation schemes are also ideal for representing heuristic knowledge in an efficient and explicit manner. (Barr & Feigenbaum, 1982, pp. 173-179)

The principle disadvantage of using procedural representation schemes is that as the system evolves it becomes increasingly difficult to modify and extend the system. Another important disadvantage of procedural representation schemes is that in most cases it is more difficult for humans to understand the knowledge that is represented in such a scheme than knowledge that is represented a declarative manner. Furthermore, since procedural representation schemes are not based on formal logic they do not provided completeness or consistency. Finally, because control information and knowledge are so closely tied in procedural representation schemes, these schemes sacrifice much of the flexibility that is achieved in declarative representation schemes. (Barr & Feigenbaum, 1982, pp. 173-179)

4. Production System

A production system is made up of a collection of IF-THEN rules (referred to as production rules or, more simply, as productions) and a simple inference mechanism. In a typical production system each production rule is a simple conditional statement that contains a set of premises and a conclusion. Each rule is an indepen-

dent, modular section of code which represents a specific aspect of the knowledge domain.

The principle advantage of employing production rules as a representation scheme is the modularity of the rules. Production systems can be easily modified and extended because existing rules can be changed and new rules can be added to the system at any time and without altering other rules. The use of production rules also greatly simplifies the process of acquiring knowledge from domain experts who are not familiar with AI techniques. The reason for this simplification is that the if-then syntax of production rule is similar to the way that many human experts structure (i.e., think about) their knowledge. Yet another advantage offered by a production rule facility is that it is relatively easy to implement an explanation facility in such a system. (Davis & Buchman, 1977)

The principle disadvantages of production systems is that it is awkward to represent non-procedural knowledge in such a system (Tanimoto, 1987, p. 130). A major problem with production systems is that the reasoning process is completely mechanical and these system only reason about the knowledge that has been explicitly defined. In addition, it is difficult to represent explicitly sequential algorithms in production rules. (Barr & Feigenbaum, 1982, pp. 189-199)

5. Structured Object Representations

Representing knowledge in the form of structured objects was first pioneered by Marvin Minsky. In his seminal article " A Framework for Representing Knowledge" he introduced the concept of "frames". The motivation for his work was a desire to develop a representation methodology that closely modeled the cognitive process. Minsky felt that the existing methods of knowledge representation were too finely grained and he proposed that knowledge is more than just a "collection of separate, simple fragments". In the appendix of the article he claims that knowledge rep-

21

resentations based upon logical formalisms "do not work" in realistic, complex domains. (Minsky, 1981, pp. 95-128)

According to Minsky "A frame is a data-structure for representing a stereotyped situation ... (and) attached to each frame are several kinds of information" (Minsky, 1981, p. 95). In the system that he proposed the individual frames are linked together into a framework based upon the relationships between the frames. The "stereotypical situations" that Minsky describes can be either concepts, physical objects, or events. The "several kinds of information" he alludes to can be considered description or attributes of the stereotypical situation. An important feature of Minsky's frame system is that both procedural knowledge and declarative knowledge can be attached to frames.

Many researchers have extended and revised Minsky's original concept. Currently most of the ideas that Minsky put forth are subsumed into the active area of research referred to as object oriented programming. The most significant concept that has been added to Minsky's theory is the distinction between classes and instances (Malpas, 1987, p. 273). In addition, most object-oriented systems explicitly support the construction of inheritance hierarchies (frameworks) based only on the "IS-A" relation. The development of frameworks based upon other relations, such as the "PART-OF" relation, is normally not explicitly supported.

The primary advantage that structured object representation schemes provides is that both procedural and declarative knowledge can be easily represented in a logical structure. This logical structure simplifies the acquisition, storage, and modification of the knowledge. In addition, this scheme supports the notion of "default reasoning" and consequently the construction of an inference mechanism can be greatly simplified.

The principle disadvantage of this representation scheme is that use of default reasoning can result in incorrect conclusions. In addition, the use of multiple inheritance, which most present day frame-based systems support, can also cause incorrect conclusions to be inferred.

D. SUMMARY

The representation of knowledge has been, and continues to be, at the core of AI research. Over the years a number of very different representation schemes have been proposed and implemented in AI systems. Each of these representation methodologies emphasize a particular aspect the of knowledge representation issue and each offers both advantages and disadvantages. In addition, each representation scheme is especially well suited at representing some forms of knowledge and less effective at representing other forms. Most complex AI systems, like ICAI systems for instance, are made up of a wide variety of knowledge types and forms. Consequently, a single knowledge representation scheme is generally not suitable for such systems. Instead a programming environment that supports multiple, integrated representation schemes should be used in the design and development of such systems.

IV. KNOWLEDGE REPRESENTATION ISSUES IN ICAI SYSTEMS

A. INTRODUCTION

We have established the importance of knowledge representation in AI in general and in ICAI systems in particular. We have also established that over the years AI researchers have developed a number of distinctly different knowledge representation schemes. Not surprisingly, the designers of existing ICAI systems have used many different schemes to represent knowledge in their systems. The following review will provide a brief description of five historically significant ICAI systems. The developers of each of these systems took a different approach to the issue of knowledge representation.

B. KNOWLEDGE REPRESENTATION IN EXISTING ICAI SYSTEMS

1. SCHOLAR

As mentioned earlier, SCHOLAR was the first Computer Aided Instruction system developed using AI techniques. It was designed and implemented by Jaime Carbonell and it provides instruction in the domain of South American geography. Its most notable feature is that it provides a "mixed-initiative" environment in which the student plays an active, rather than a passive, role in the instructional process. A description of the system was first published by Carbonell as part of his Doctoral dissertation in 1969. He later extended and modified, the system in conjunction with his associates when he was employed at Bolt, Beranek, and Newman, Inc. (BBN). (Barr & Feigenbaum, 1982, pp. 236-241)

24

When Carbonell began work on SCHOLAR his intention was to implement a sophisticated instructional system with the functionality that is provided by all four components of the generalized ICAI model. In order to provide this functionality, he needed to represent a significant amount of knowledge in various forms. He elected to use a semantic network as the principle representation scheme in his system. The semantic network which he constructed serves as the basis for the representation of nearly all of the knowledge in the system.

The domain knowledge in SCHOLAR is primarily a set of declarative facts. As a result, Carbonell was able to construct a detailed semantic network which effectively represented the domain knowledge (see figure 2). The nodes in the network represent physical objects (for instance, Argentina) or concepts (for example, latitude) and the arcs in the network represent the relationships between the nodes (for example, super class). By employing a semantic network Carbonell achieved a high degree of flexibility that allowed him to use the knowledge in a variety of ways. Because of this flexibility he was able to use the network as the basis for the other components in the system. (Carbonell, 1970, pp. 190-202)

Carbonell also used the semantic network as the basis for his model of the student. In the system the student is initially modeled in terms of the full semantic network (i.e., the student is assumed to "know" everything about the domain before instruction begins). As a typical student uses the system, however, he normally demonstrates that he possesses one or more misconceptions related to the subject matter. The system diagnoses these misconceptions and uses this information to update the student model. It updates the student model by modifying the nodes and/or links in the semantic net which represents the student's knowledge of the subject matter. (Wegner, 1987, p. 37)
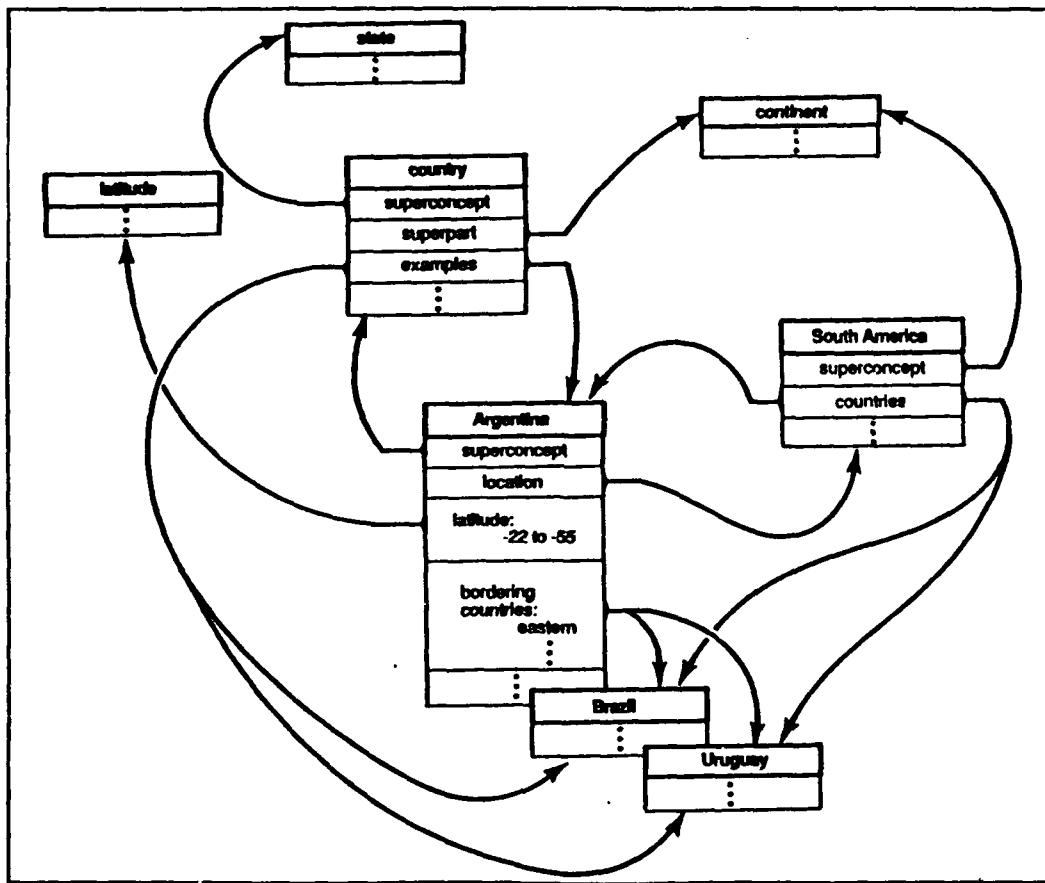
Figure 2 - Example Semantic Network from SCHOLAR (Wagner, 1987, p. 32)

The tutoring model in SCHOLAR also makes extensive use of the semantic net to perform their functions. The tutoring model in SCHOLAR encapsulates a Socratic dialogue tutoring strategy. A Socratic dialogue is an interaction between a tutor and a student in which the tutor assists the student in the resolution of his misconceptions by asking well-timed and pointed questions (Kearsley, 1987, p. 17). The tutoring model, in SCHOLAR, provides Socratic tutoring by responding to the misconceptions that are reflected in the semantic network which models the student.

SCHOLAR's communication model also uses the semantic network extensively to perform its function. The communication model takes advantage of the manner in which facts are structures in the semantic network. It uses the structure of the network to formulate both natural language questions for the student and natural language responses to the student's questions. The communication model interprets the student's questions by looking for semantic clues in his input. It uses these clues to determine the meaning of the student's question and extract information from the semantic network.

2. BUGGY

BUGGY is a diagnostic system which diagnoses student misconceptions about the procedures involved in doing subtraction. The system was designed in the late 1970's by two leading authorities in the ICAI field, John Seely Brown and Richard Burton. DEBUGGY was originally on off-line system, however, it was later upgraded into an on-line, interactive system called IDEBUGGY. Although, all four components of a generalized ICAI are present in IDEBUGGY; the knowledge model and the student model are the most interesting. These two models are especially interesting for two reasons. First, because of the theory upon which these models were based, and second, because of the unique representation scheme that was used to capture the knowledge in both models. (Barr & Feigenbaum, 1982, p. 279)

The theory that the designers of this sytem used was that students make errors in the performance of procedural skills not as a result of an inability to follow procedures, but instead, because they have learned one or more faulty sub-procedures. The designers developed a cognitive model based on this theory which they called the "buggy model." Their buggy model served as the basis for both the knowledge model and the student model, in DEBUGGY and its descendants (Burton, 1982, pp. 157-187).

27

The designers of DEBUGGY used a procedural network as the knowledge representation scheme to implement their theoretical model. Consequently, their domain knowledge model consists of a detailed procedure which performs the subtraction operation. They broke this "expert" procedure into a lattice of finely grained sub-procedures in which each sub-procedure encapsulates a specific subskill of subtraction. Burton defines a subskill as "any isolatable part of the skill that it is possible to mislearn" (Burton, 1982, p. 177).

In the system, the student model was not a conventional "overly" model (in which the student is assumed to possess a subset of the expert's knowledge). Instead, the model of the student's knowledge state was based on the semantic network. The use of a procedural network allows for the student's knowledge of the subject matter to be easily modeled. The reason for this ease of modeling is that for procedural skills, a student can always be assumed to possess a variation or perturbation of the procedural network which represents the expert's knowledge. (Burton, 1982, pp. 157-187)

The original system worked in the following manner. The student takes a test (off-line) which consisted of 12 multi-column subtraction problems. His answers are provided to the system and if errors are detected the system attempts to infer the cause of the errors (i.e., diagnose the student's "bugs"). The system makes this inference by replacing one or more of the sub-procedures in the procedural network with "buggy" procedures. The system would continue making substitutions until the system could either generate results that were "close" to the answers provide by the student. The specific modified procedural network that produces the set of answers which were the "closest match" is an accurate model of the student's knowledge of the subtraction operation. Each of the "buggy" procedures in this student model represents specific misconceptions that the students has demonstrated in his work.

28

3. GUIDON

The subject matter addressed in GUIDON is infectious blood disorders. The system was designed and developed by Clancey and others at the University of Stanford in the late 70's and early eighties. The system went through numerous revisions and extensions until work was discontinued on the project in 1985. The most notable characteristic of GUIDON is that it was the first tutoring system which was based upon an existing expert system. (Clancey, 1986, p. 40)

The domain knowledge model for the original GUIDON system was the rule-based expert system, MYCIN. Consequently, the primary representation scheme for the domain knowledge was a production system. Clancey took full advantage of the characteristics of this representation scheme to enhance the tutorial effectiveness of the GUIDON system. He took advantage of the fact that structure and syntax of production rules makes them "human readable" and consequently, very useful in the instructional process. In addition, because production rules were used the knowledge domain was modularized into a set of independent and finely grained concepts. Production rules were such an effective representation scheme that they were used as the basis for the other three components in the system.

The knowledge in the tutor model, for example, is a represented by a collection of production rules which Clancey calls "t-rules". These t-rules, which numbered over 200, captured the procedural knowledge associated with GUIDON's tutoring strategy and pedagogical style. These t-rules can be best described as meta-rules or meta-knowledge because they reasoned about, and work on, other rules in the system. (Clancey, 1986, p. 41)

The student model in GUIDON also makes extensive use of both the MYCIN production system and other production rules. To model the student, Clancey used an overlay model in which the system gave the student "credit" for partial solutions.

GUIDON is able to assign credit in such a manner because of the following features of the system: it generates an "approved solution" to each diagnosis problem before it presents it to the student and it uses another set of production rules (really a subset of the "t-rules") to compare the system's solution with the student's partial solutions. The system also keeps track of the student's response history to better determine his current knowledge state.

The communication model in GUIDON, although not implemented as a separate module, is also based primarily on a production rules. The communication model can not be considered a separate module because it is tied directly to the tutor model. The "t-rules" which serve both to guide the student through the instruction, and diagnose his current knowledge state, also serve to determine how information is presented to the student and how student input are interpreted by the system. These rules control the natural language discourse with the student.

4. Power Distribution Tutor

In the mid-1980's J. R. Bourne and his contemporaries at the Center for Intelligent Systems at Vanderbilt University, implemented two ICAI systems which provide instruction in engineering domains. The two specific domains which are addressed by these implementations are: (1) a hot water heating system, and (2) a power distribution system. Both systems have a number of interesting features, however, in this section we will focus on the power distribution system tutor (PDST). (Bourne et al, 1988, pp. 213-226)

The PDST system employs a sophisticated, interactive, graphical model of a power distribution plant. It is a "complete" tutoring system in which all four components of the generalized ICAI are present. An important difference between the ICAI's discussed earlier and PDST is the approach that its designers took with the respect to the knowledge representation issue. Instead of depending on one primary

representation scheme, the designers of PDST employed a variety of representation schemes in their system. The PDST designers used both structured object and production system representation schemes throughout their application.

The domain knowledge model in PDST is a sophisticated model-based simulation of a power distribution system. The representation scheme that was used to develop the simulation was a structured object representation. The domain knowledge lent itself well to this form of representation because the system is made up of numerous components, each of which can be modeled as distinct objects. Since each component is modeled as an object, the static, declarative information concerning each object and its relationship with other objects were stored together in one frame. In addition, since many of the objects in the system are of the same type or class the system designers were able to organize the objects into an inheritance hierarchy. Still further, the dynamic, procedural information that is associated with each component in the system was attached to the component's frame. The dynamic behavior of an object in this application is normally its response to changes in the system. (Bourne et al, 1988, pp. 213-226)

The tutoring model in PDST, referred to by the system's designers as the Strategist, uses a production system as its primary representation scheme. This scheme was selected because it was determined that the knowledge which must be captured by the tutor model is best represented in the form of production rules. The system's designers were attempting to capture human expertise in this component of their system and production rules are especially well suited for this purpose.

The representation scheme that was used for the student model in PDST was also a production system. The PDST designers implemented a "bug" model to model the student's current knowledge state. They took advantage of the modular nature of

rules and consequently each potential "bug" that the designers identified was captured in a single rule.

The communications model in PDST is a sophisticated, interactive graphical interface. The system's various components are displayed graphically and the user can affect the state of the system by pointing at, and selecting, components with a mouse. The designers of PDST used a structured object representation to capture the knowledge in the communication model. This representation scheme was selected because each of the graphical images that are displayed represents an object in the system.

C. SUMMARY

Knowledge representation is clearly one of the most significant issues in the design of ICAI systems. ICAI systems are generally very complex and consequently the designer of such systems must represent many different types and categories of knowledge within their systems. They must represent the following different types of knowledge within the four components of their systems domain knowledge, cognitive knowledge, interface knowledge and pedagogical knowledge. In addition, they must represent knowledge which falls into the following categories: declarative knowledge, procedural knowledge, and meta-knowledge.

The preceding review of existing ICAI systems should give the reader an appreciation for the variety of approaches that ICAI designers have taken with regard to the issue of knowledge representation. In the SCHOLAR system, an associational-based representation scheme, a semantic network, was employed as the principle representation scheme. In BUGGY, a procedural-based representation scheme was used throughout. In GUIDON, a production system was employed to capture all of

the knowledge in the sytem. Finally, the PDST system used both a production system and a structured object representation schemes.

V. APPLICATION DEVELOPMENT ENVIRONMENT

A. INTRODUCTION

We have designed two ICAI systems in the Knowledge Engineering Environment (KEE) on a Texas Instruments Explorer workstation. The development environment is described below for the reader to gain a basic understanding of the tools and facilities we used in the development of both applications. KEE terminology is also important to know for the later descriptions of the applications.

B. TEXAS INSTRUMENTS EXPLORER

The Explorer system is an advanced, single-user workstation that provides extensive support for developing knowledge-based applications and for rapid prototyping. It offers a flexible, interactive, and highly productive programming environment with run-time support for efficient execution of sophisticated programs. The explorer system offers the user the following benefits:

(i) Natural manipulation of symbols that can represent abstract concepts using the Lisp language in a high-speed hardware architecture especially designed for symbolic processing;

(ii) Ability to create large programs without concern for memory allocation and deallocation, taking advantage of the system's efficient incremental garbage collection and virtual memory management of up to 128 megabytes of virtual memory; and

(iii) Transparent sharing of resources with a variety of computer systems through networking facilities. (Texas Instruments Incorporated, 1988, p. 1-1)

The specially designed and microcoded Explorer system processors directly implement the software run-time environment, providing fast execution of large programs

without sacrificing the dynamic nature of Lisp. The Explorer system is written entirely in Lisp and supports Common Lisp.

C. KNOWLEDGE ENGINEERING ENVIRONMENT

The Knowledge Engineering Environment (KEE) provides knowledge system developers with a set of programming tools and techniques for building applications to represent, analyze, and reason about knowledge, and to construct application interfaces. It is built upon Lisp and supports object oriented programming. The following sections describe features of KEE so the reader will have a basic understanding of KEE terminology presented in follow on chapters.

1. Objects

Since KEE is based upon the object-oriented programming paradigm, it contains a flexible and expressive frame system for representing and managing knowledge. The basis of the frame system is the object. Objects are discrete data structures used to represent real world objects. Objects can be grouped together in class-subclass-instance hierarchies to logically organize objects in an application and to take advantage of inheritance. An object contains slots which describe attributes about the object. Slots are used to represent two kinds of information: descriptive and procedural. They can store numerical and textual data about the object as well as more complex information such as graphics structures, references to other objects, and procedural programs that can execute Lisp functions or begin rule chaining. A slot containing procedural information is called a method. Methods are procedures with local state that specify the behavior of the object in response to messages sent to the object.

35

2. Inheritance

Designers can describe a system by creating objects that act as templates for whole classes of objects. When an object is designated as a "child" of one or more objects, it inherits slots and default information from its ancestors. Inherited information can be added to or modified only as needed to differentiate the child object from its parents.

3. Rules

One way of presenting unordered, declarative knowledge in an ICAI system is through rules The rules either state new facts to the system and let it determine the consequences of these facts (forward chaining), or ask the system about a particular goal and let it determine whether or not that goal can be deduced, given the facts that the system already knows (backward chaining). In KEE, the reasoning capability on rules is supplied through tools called Rulesystem3 and TellAndAsk. Rulesystem3 provides forward and backward chaining capabilities and the means of controlling and debugging them. TellAndAsk integrates these capabilities with the larger frame system.

In KEE, rules and rule classes are represented as objects whose attributes are defined by Rulesystem3. Each rule is an object, and each rule object is a member of at least one rule class. In addition, KEE allows users to partition rules by grouping them into rule classes. This increases the system efficiency by reducing CPU time for pattern-matching routines and user maintenance time. Since rules and rule classes are represented as objects, the KEE frame system and interface is available for creating and manipulating rules and rule classes. Additionally, since they are KEE objects, rules can be manipulated and reasoned over like any other KEE object.

KEE provides two types of rules for the designer: deduction rules and action-taking rules. Both types are needed depending upon whether the user requires rules

that perform deduction and rules that take some kind of action on the system objects. Deduction rules are pure theorem-proving rules. They establish dependencies between a conclusion and its premises. If the rules produce a contradiction to the set of facts that the system started out with, the original facts are inherently contradictory or the rules are in error. Action-taking rules, on the other hand, change the knowledge base of the application, for example, making an additional assumption or taking an action. Depending on where the change in the assumption set takes place, action-taking rules are either *same world action rules* or *new world action rules*. (Intellicorp, Inc., 1988, p. 1-3)

4. Truth Maintenance System

Dependencies can be set up between facts in a knowledge base, such that a particular fact is always true when one or more other supporting facts are also true. If a fact or assumption is no longer believed to be true, KEE's Assumption-based Truth Maintenance System (ATMS) makes sure any facts contingent upon the belief are retracted from the knowledge base.

5. KEEworlds

KEE provides a multiple worlds facility called KEEworlds that enables a user to assert a collection of facts into a single context, called the background, which can be used as a basis for the creation of additional worlds containing somewhat different facts and assumptions. KEEworlds is useful for modeling and exploring different hypothetical situations that might arise in a knowledge base. A new world represents an alternative state or new context of a knowledge base.

6. Backward and Forward Chaining

Rules designed in KEE may be linked together into chains, by matching the premise of one rule to the conclusion of another rule or by matching the conclusion of one rule to the premise of another rule. In this way, a rule can be used to help prove

or derive another rule. Backward chaining (also called goal-driven reasoning) attempts to establish the conclusion of a rule by seeing if the rule's conditions can be established using facts in the system or by triggering other rules. Forward chaining (also called data-driven reasoning) starts with a set of facts and matches them to the premises of a rule. If the premises are satisfied, the rule's conclusions are added to the knowledge base and may be used to draw additional facts.

7. TellAndAsk

TellAndAsk is an English-like language that can be used with KEE. It can be used for a variety of purposes such as interacting with knowledge bases, writing rules, creating justifications, and putting facts into worlds. TellAndAsk is a language with a syntax similar to English and provides a way of making statements about: relationships between units, values of slots, and values of facets.

8. KEEpictures

The KEEpictures environment is an object-oriented graphics toolkit that enables designers to construct images or graphic systems, interfaces, and end-user applications. Its intended uses range from simple graphical displays to animation. KEEpictures provides a set of standard picture classes for picture construction. These include arcs, axes, box.strings, bitmaps, circles, dials, lines, and rectangles. Each of these primitives can be shaped, enlarged, shrunk, and altered in a variety of ways. Pictures can be combined to form larger, composite pictures. Users can also draw their own pictures using bitmaps.

9. ActiveImages

KEE's ActiveImages package provides a variety of graphic displays for both viewing and modifying slot values in KEE objects. These graphic images can be integrated into an application as an attractive interface to the system. They also can be used during application development as a convenient display aid for debugging pro-

grams. There are over twenty different images, including bargraphs, histograms, thermometers, switches, push buttons, and plots.

10. Active Values

Active values are KEE objects which allow a designer to specify actions to happen when slots are accessed or modified. They provide the facility to cause side-effect behavior when accessing or modifying data in a knowledge base. By attaching an active value to a slot, the designer can specify that a particular action should occur every time that slot's value is accessed or modified.

D. KNOWLEDGE REPRESENTATION IN KEE

KEE supports various knowledge representation schemes including structured object representations, procedural representations, production system, associational representations, and formal logic-based representations.

Since KEE is based upon a frame system for representing and managing knowledge, it is obvious that KEE supports objects as a form of structured object representations. Both declarative and procedural knowledge can be represented in slots of objects.

Procedural knowledge can be represented in KEE in a procedural representation scheme. An object created solely for representing a procedure contains only one or more methods, no slots with descriptive information. Such an object explicitly represents procedural knowledge.

KEE explicitly supports the use of production rules through its Rulesystem3 knowledge base. Procedural knowledge can be represented by production rules. Reasoning can be accomplished by backward or forward chaining.

Semantic networks can be created, although KEE does not explicitly support their creation. Any relationships within knowledge in an associational representation schemes must be designed by the user using slots and methods in objects.

KEE's TellAndAsk language provides a means of representing knowledge in a formal logic-based representation scheme. Knowledge is represented as declarative facts in the form of English-like statements called well-formed-formulas (wff). Wff's can be added to, retrieved from, or deleted from a knowledge-based system. Wff's with or without variables evaluate to true, false, or unknown. KEE also supports the use of deduction rules to represent generalized dependencies between facts, to create constraints, and to control search.

VI. MAP READING TUTOR

A. INTRODUCTION

The Map Reading Tutor (MAPTR) is an ICAI for teaching U.S. Army soldiers the common skills tasks required for proficiency in reading and using a military map. Such skills are necessary for conducting military operations. A potential student of the system can be unfamiliar with the material or proficient at some or all of the tasks which the system teaches. The system behaves as a tutor who teaches a lesson plan of tasks. The lesson plan is developed either from student input or based upon the system's diagnosis of the student's knowledge state. Text and graphics are presented to the student in a predetermined sequence for each concept contained in the task. At the completion of a task, the tutor tests the student on his understanding of the material. Remedial training is provided for tasks which the student is deficient.

B. THE TASK: MAGNETIC AZIMUTH

The system currently contains knowledge to teach the task: magnetic azimuth. The azimuth is the most common military method to express direction. Determining and using a magnetic azimuth is a fundamental skill without which it would be difficult to plan and conduct military operations. The instruction provides the soldier with knowledge about azimuths, the three different norths, the declination diagram, and the conversion of azimuths from grid to magnetic, and vice versa.

The instruction begins with the definitions of the three norths: true north, magnetic north and grid north. The soldier must understand that magnetic readings obtained by a magnetic compass indicate the direction to the north magnetic pole; while grid north

41

is the north that is established by using the vertical grid lines on the military map. Next, the declination diagram is introduced. The declination diagram is a part of the marginal information on all military maps. It shows the angular relationship among the three norths. The soldier is then taught the concept of azimuth conversion using the grid-magnetic (G-M) angle found in the declination diagram. The G-M angle is the angular size that exists between grid north and magnetic north. Mastering this concept is essential for plotting a magnetic azimuth on a map or determining a magnetic azimuth from a map.

C. MAPTR ARCHITECTURE

MAPTR has the same general components of a typical ICAI system. The functions of each component are presented below.

1. MAPTR Domain Knowledge Model

The domain knowledge model contains knowledge about the concepts of azimuths, the three norths, the declination diagram, and azimuth conversion. It is used in the generation of lesson plans and to evaluate the soldier's performance. The concepts are captured in the form of text and graphics. Training modules contain multiple concepts in a predefined sequence for instruction. The knowledge is programmed for three training modes: review, train, and retrain. Each concept is represented in three different ways to satisfy a soldier's requirement to review, train, or retrain the subject. Additionally, main concepts and tasks are evaluated. The domain knowledge model also includes the proficiency thresholds of each task for the three training modes.

The domain knowledge model is organized as a taxonomy of tasks. Each task has subtasks, supertasks, and a training module associated with it. A training module consists of the major concepts of a task. The major concepts of a training module

42

are represented as submodules. A training submodule contains the declarative or procedural knowledge associated with a particular concept. The knowledge representation scheme used in this system to represent both types of knowledge in the knowledge model is a form of structured object representations called objects. Additionally, we incorporated the objects into a form of associational representation scheme called a semantic network.

The taxonomy of tasks is a hierarchy of objects containing many slots. Each object contains slots identifying the task's supertasks and subtasks, the prerequisites for learning the task, the time period requiring review or training of the task, the proficiency threshold, and the training module for the task. The supertasks, subtasks, and prerequisites slots of a task add a relational attribute to the object. A task may be a supertask, subtask, and a prerequisite of another task. The objects that represent the tasks are considered nodes of the semantic network. The slots contain the associative links to other objects and are considered the arcs of the network.

The purpose of a training module is to organize the concepts of a task depending upon the training mode: review or train. It embodies the sequence in which the concepts of a task are presented. The presentation sequence of the main concepts of a task depends upon whether the soldier requires reviewing or training. A training module consists of a group of objects representing each major concept in the task. The objects have two slots: review and train. When the training mode is determined, the appropriate slot of each major concept is accessed to acquire the correct set of training submodules.

The training submodules are a group of objects that contains the knowledge of a concept. The text and graphics that explain and illustrate a concept are different depending upon whether the soldier is reviewing or learning the concept. Hence, there are two objects for every concept, one for each training mode. Each object has a slot

43

containing a list of objects that contain the text and graphics associated with the concept.

A concept may be composed of three types of knowledge: declarative, procedural and heuristic. Examples of each type are shown in Figures 3 and 4. The knowledge model contains a group of objects that correspond to all of the declarative, procedural, and heuristic knowledge in the task. Each object has two specific knowledge slots for the concept which it encapsulates. One of the slots contains a reference to a graphical representation of the fact; the other slot contains the explanation of the fact.

Facts:

- There are three kinds of norths: true north, magnetic north, and grid north.

- Grid north is the north that is established by using the vertical grid lines on the map.

- The angle between grid north and magnetic north is called the Grid-Magnetic Angle.

- All military maps contain a declination diagram which illustrates the relationship between the three kinds of norths.

- The lensatic compass is used to measure the base direction of magnetic north.

Figure 3 - Declarative Knowledge

Procedure:

Convert a magnetic azimuth to a grid azimuth:

1. Determine the type of G-M angle on the map.

2. If an east G-M angle then add the value of the G-M angle to the magnetic azimuth.

3. If a west G-M angle then subtract the value of the G-M angle from the magnetic azimuth.

4. If the resulting grid azimuth is greater than 360 degrees then subtract 360 degrees from it.

5. If the resulting grid azimuth is less than 0 degrees then add 360 degrees to it.

Figure 4 - Procedural and Heuristic Knowledge

For example, we created an object called FRAME.THREE.NORTHS to teach the concept about the three kinds of norths. The object has two slots, MYPIC-TURE and MYTEXT. Figure 5 shows the value of the slot MYTEXT which describes the concept. Figure 6 shows the value of the slot MYPICTURE which is a pointer to an image created to illustrate the concept. Two additional slots contain information for use during student evaluation of the task. A slot called RE-SPONSE.LIST contains the possible solutions to a problem which is posed to the soldier during an evaluation. The solutions are displayed in a menu for the soldier's selection. The other slot is called CORRECT.RESPONSE and contains the solution to the corresponding problem. The use of these two slots is explained in the section which describes the tutor model.

45

When we define azimuth we say it is measured from the base direction of north. In this world of ours we have three norths and you see them graphically represented in the declination diagram of your map. Check any military map and you'll see that the three norths are:

- TRUE north - a line from any position on the earth's surface to the geographic north pole. True north is symbolized by a line with a star at the apex.

- MAGNETIC north - the direction in which the magnetic arrow of the magnetic compass points. Magnetic north is symbolized by a half arrowhead at the apex of a line.

- GRID north - the north that is established by grid lines on the map. Grid north is symbolized by the letters GN at the apex of a line.

Figure 5 - Text


True
North

Magnetic
North

Grid
North
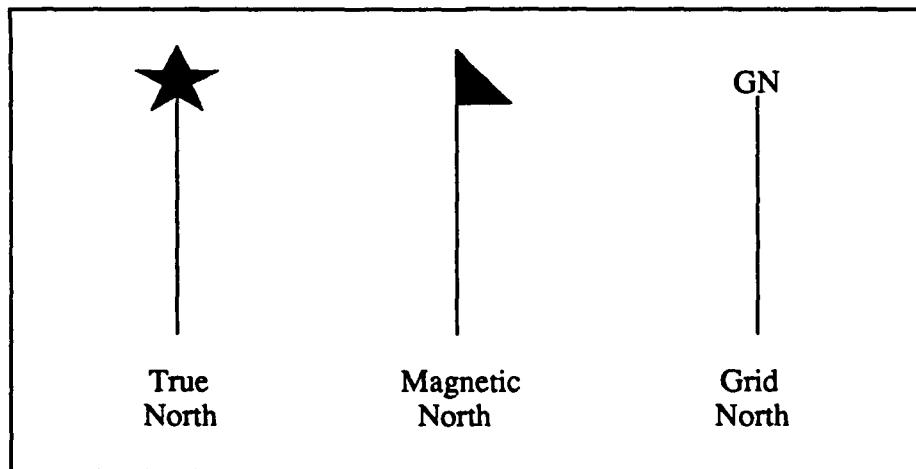
Figure 6 - Graphics

All declarative knowledge is represented in this manner. Procedural knowledge is decomposed into primitive operations. Each operation is again represented using a form of structured object representations called objects. A procedure then is presented as a sequence of primitive operations.

2. MAPTR Tutor Model

The tutor model manages what instructional material the system presents and how and when it should present it. Using the student model and knowledge model, the tutor model adaptively constructs an individual lesson plan for teaching the soldier. First, it assesses a soldier's knowledge state by comparing the proficiency threshold of a task to the soldier's proficiency of that task and then checking the date of proficiency. From this, the tutor model determines the tasks to teach and the training modes to use for the tasks. Then it presents displays of information and graphic instruction screens covering the tasks, concepts, and evaluations in the lesson plan. Finally, the tutor model updates the student model upon completion of evaluations.

The type of knowledge which the tutor model represents is procedural knowledge. We used a procedural representation scheme to represent the procedural knowledge. Many of the objects described in the knowledge model contain methods that represent p.....s of the procedural knowledge. Each object that represents a task in the taxonomy contains a method called TEACH that formalizes all of the tutor's procedural steps for teaching that task. The procedure in the TEACH method contains many subprocedures that are reflected as methods of other objects in the knowledge model. Such methods include getting the proficiency thresholds of the task for each training mode; obtaining the appropriate training modules depending upon the assessed training mode; and displaying text and graphics.

The TEACH method is a recursive Lisp function that is a method of every task in the knowledge model. The procedure is described as follows. When a task is se-

lected to be taught, either by the soldier or the system, the TEACH method is invoked. The method retrieves the training modules, prerequisites, and subtasks of the task for assessment. If there are prerequisites then the method requests from the student model the soldier's proficiency in each task in the prerequisite list. If the soldier is not proficient or needs to review any task on the prerequisite list then the TEACH method activates that task to teach the soldier; otherwise, the method checks the list of subtasks. The method repeats the same actions on the subtask list as with the prerequisite list. If no subtasks exist or the soldier is proficient in the subtasks then the method checks the training module list for the current task. The method requests from the student model the training mode in which to teach the soldier. Then the TEACH method activates a method in the appropriate training submodule to show its list of objects that contain the instructional displays. These objects contain a method that displays its text and graphics to the screen.

The last item on the training module list is an evaluation of all the material covered in the training module. Soldier input during an evaluation is through a menu system. The tutor model compares soldier responses to the value of the CORRECT.RESPONSE slot of the appropriate display object, assesses the soldier's proficiency, and provides feedback. If the soldier is proficient then the tutor tells the current soldier's student model to update itself; otherwise, the tutor activates a method in the training submodule for retraining to show its list of display objects. After retraining is completed, the tutor repeats the evaluation process. If the soldier is still deficient, the tutor records the information for later feedback.

3. MAPTR Student Model

The student model reflects the soldier's proficiency in the task. It receives performance evaluations with respect to the task the soldier is expected to acquire

from the tutor model. These evaluations are permanently stored for analysis by the tutor model to determine lesson plans.

The student model consists of declarative knowledge about the soldier's state of knowledge. A form of structured object representations called objects is used to represent the declarative knowledge. When a new soldier enters the program, three objects are created using his social security number (SSN) that form a composite object. Figure 7 illustrates an inheritance hierarchy of several KEE objects which form a student model for two students. The objects containing the SSN, 111-11-1111, form a composite object that constitutes the student model for the soldier with that particular SSN. These three objects and their various slots will be described in detail.

The first object called 111-11-1111 contains five slots. Two slots contain pointers to the other two objects in the composite object, 111-11-1111.PROF.DATA and 111-11-1111.ADMIN.DATA. The other three slots are methods for assessing the soldier's knowledge state, for recording the soldier's proficiency during a training session, and for deleting a student record.

The second object called 111-11-1111.ADMIN.DATA currently contains three slots for recording administrative data: FIRST.NAME, LAST.NAME and SEX. This object could contain as many slots as needed to record all administrative information required.

The third object called 111-11-1111.PROF.DATA currently contains a large number of slots, most of which represent each task in map reading. These slots contain a value representing the soldier's current proficiency in the respective task. Four

49

additional slots are methods for initializing this object at creation and for assisting the methods in the 111-11-1111 object.
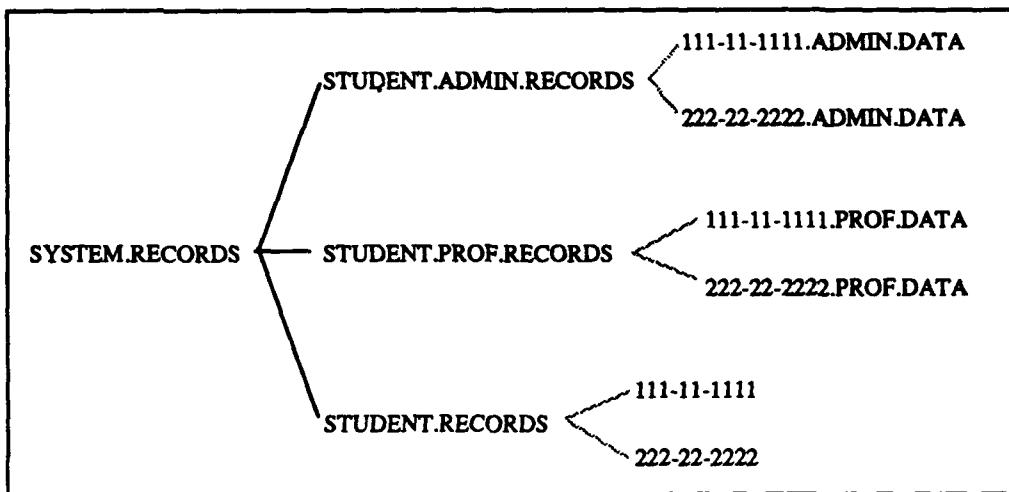
Figure 7 - MAPTR Student Model

4. MAPTR Communication Model

This model provides an interface to the soldier for instructions, text and graphics, system responses, and soldier input. Windows and pop-up menus are provided. All student input is accomplished by a mouse. The communication model for MAPTR contains procedural knowledge. We used a procedural representation scheme to represent the procedural knowledge.

The interface consists of an input window, an output window, viewports, and menus. Figure 8 illustrates a typical screen during a tutoring session. We designed window and menu methods that programmatically created the input and output windows and the menus. We created a standard KEE typescript window that is displayed for the user to input administrative information and for the system to display system instructions. We created a standard KEE output window that displays all text for domain knowledge instruction. We also created numerous viewports for dis-

playing graphical images of concepts in support of the textual instruction. We also created pop-up menus to accept input from the user. The soldier can select the tasks to be taught or enter an answer during an evaluation using these menus. The KEE system contains built-in functions for implementing the window and menu methods we designed.
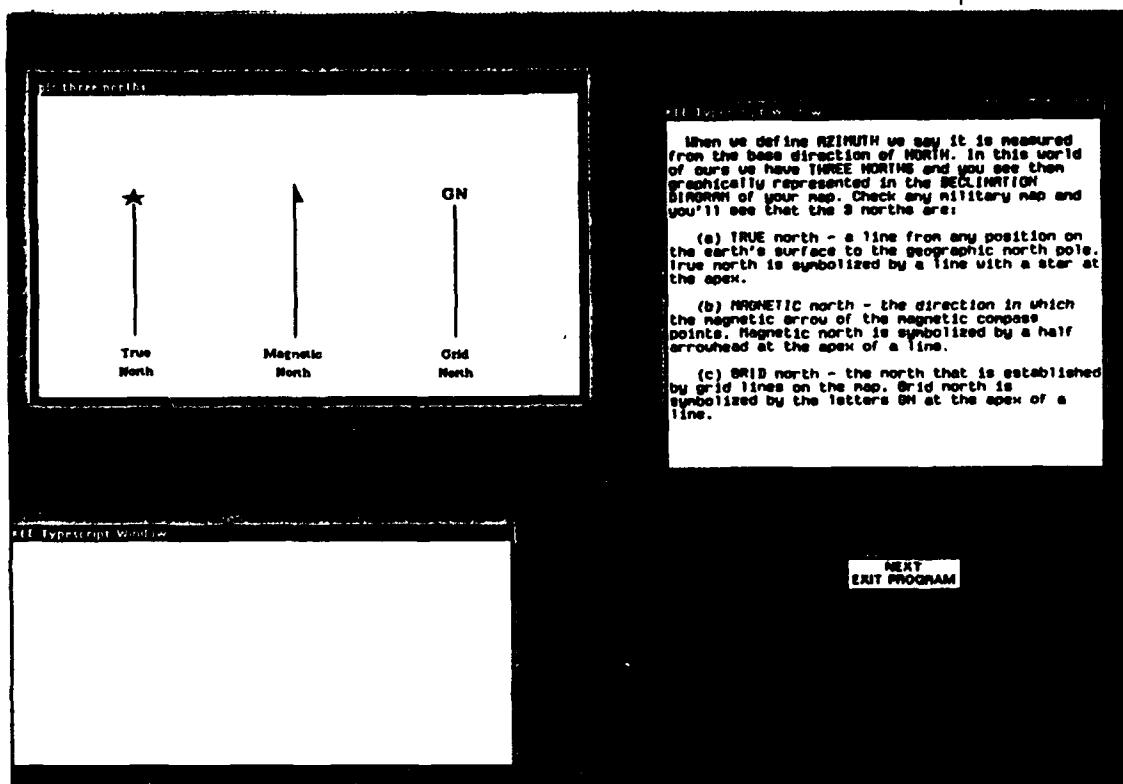

Figure 8 - MAPTR Interface

D. SAMPLE TUTORING SESSION

The interactions between the four models is described through the course of a typical tutoring session. When a soldier logs into the system and provides his SSN, the

51

student model checks for a student record matching the SSN. If one does not exist, the student model creates a composite object representing the new soldier's student record. The soldier is then presented a sequence of menus for the soldier to choose the desired task to be taught. The tutor model interacts with the knowledge model and student model to determine (1) if any prerequisite tasks need to be presented, and (2) the mode of instruction. The knowledge model interacts with the communication model to display the text and graphics that constitute the instruction for the task and to present an evaluation at the conclusion of the instruction. Through the course of instruction, the tutor model checks for required subtasks. If subtasks exist, the tutor, student, and knowledge models repeat the process of determining the mode of instruction and presenting the instruction and evaluation. From an evaluation, the tutor model determines if the soldier is proficient at a task. If not, the tutor model changes the instruction mode and activates the knowledge model to retrain the soldier. Otherwise, the soldier can chose another task to be taught. Screen displays of a sample tutoring session is provided in Appendix B.

E. EXTENSIONS

Additional tasks which a soldier must know to be proficient at reading a map can easily be incorporated into MAPTR. The current taxonomy of tasks is structured to include all of the additional tasks. The domain knowledge model could be expanded to include the text and graphics needed for instruction of additional tasks. The student model could be updated to also include the additional tasks. The TEACH method in the tutor model would execute the tutoring process for additional tasks without any modification. With the inclusion of all tasks, MAPTR would be a suitable ICAI system for assisting trainers in the instruction of map reading skills to soldiers.

52

VII. PILOT EMERGENCY PROCEDURE TUTOR

A. INTRODUCTION

The Pilot Emergency Procedure Tutor (PEPTR) is an ICAI for teaching F-4 pilots how to detect, analyze, plan and perform appropriate actions to deal with on-board emergencies. It is designed to be a training assistant for pilots to use in maintaining their skills at solving in-flight emergencies. PEPTR has two modes of operation: a reaction mode and an instruction mode. In the reaction mode, the student can freely change the state of the aircraft to learn what in-flight emergencies are caused by various aircraft states and to learn which changes lead to a solution. While in the instruction mode, the student engages in a tutoring process that provides instruction and evaluation of a predetermined in-flight emergency. PEPTR provides the user with an interface containing switches and dials that represent aircraft systems and are used to define the initial state of the aircraft or change its state depending on the mode of operation. The expert model in PEPTR is invoked either by the system or the user depending upon the mode of operation to diagnose and determine the correct solution to an in-flight emergency. The expert model in PEPTR performs diagnosis of the situation utilizing rules that define aircraft system behavior. PEPTR checks the various switches and circuit breakers that are accessible to it and creates a list of actions that should be performed in order to overcome the emergency situation. Execution (by the user) of the advised actions involve altering switch settings. These actions will affect the state of the aircraft and may lead to resolution of the problem.

53

B. F-4 DESCRIPTION

The F-4 fuel system consists of 5 fuel tanks: a fuselage tank (consisting of 6 separate cells with approximately 8900 lbs of total capacity), an internal wing tank (consisting of 2 separate cells with approximately 4300 lbs total capacity), two external wing tanks (2500 lbs capacity each), and an external centerline tank (with 4000 lbs capacity). The fuel is fed to the two aircraft engines from the integral fuselage tank, which in turn is fed from other tanks until they become empty.

The fuel to the fuselage tank is transferred using air pressure that is provided by the engines. Except for certain situations (e.g., take-off and landing) the fuel tanks are pressurized and the fuel is transferred.

The fuel flow to the fuselage tank is regulated by the tank's fuel level valves in such way that, as long as there is fuel in other tanks, the fuselage fuel tank is kept full. For different reasons (e.g., gravity center considerations, landing stress, etc.), during most of flight cycles internal wing and external fuel tanks will not transfer fuel to the fuselage tank in concurrent fashion. Their fuel transfer is determined by an External Tank Transfer Switch that is operated directly by the aircraft pilot. The pilot is provided with additional switches (e.g., Fuel Dump Switch) to control the operation of the fuel system in normal and emergency conditions.

The F-4 aircraft is equipped with an in-flight refueling system that allows mid-air refueling of the aircraft. This system uses the same fuel lines and valves as the regular fuel system, and it allows selective refueling of internal and external fuel tanks. Although the F-4 fuel system is designed to minimize pilot's work-load under normal conditions, it requires some "switch toggling" for its regular operation, and more effort for diagnosing and rectifying fuel system malfunctions.

C. THE TASK: FUEL SYSTEM EMERGENCY PROCEDURES

PEPTR is designed to deal with malfunctions of several aircraft systems (e.g., fuel, hydraulic, electrical, engine, etc.); however, the current scope of the system deals only with some of the fuel system failures and emergencies. The student learns about the fuel system components and their operations and how the state of the aircraft affects fuel system emergencies. Fuel system emergency procedures are decomposed into four categories: *fuel boost pump failure, fuel transfer failures, failed open defuel valve during air refueling,* and *air refueling fuselage tanks.* Each category may comprise of one or more emergency conditions which contain a procedure for correcting the problem. For example, the category, *fuel transfer failures,* contains four emergency conditions, one of which is titled *internal wing fuel fails to transfer.* This emergency condition contains the following procedure that the pilot must follow to correct the problem.

> Check the following switch and circuit breaker positions:
> 1. External transfer switch - OFF
> 2. Internal wing transfer switch - NORMAL
> 3. Refuel probe switch - RETRACT
> 4. Internal wing transfer control circuit breaker - IN
> 5. To ensure wing tank pressurization:
>
>   Wing transfer pressure switch - OVRD TRAN
>   (continue in level flight for 30 seconds)

This procedure identifies what switches apply to this emergency condition and the correct positions for each switch. If the pilot diagnosis this emergency condition, then he must follow this procedure to determine if the cause is incorrect switch positions.

If so, then he must correct the switch positions in order to eliminate the emergency condition. A student can be tutored on this emergency condition and all other emergency conditions that comprise each category. Additionally, single or multiple emergency conditions can be presented to the student.

D. THE ARCHITECTURE

The major components of PEPTR are also the same as the general components of an ICAI system. The functions of each component are presented below.

1. PEPTR Domain Knowledge Model

The domain knowledge model consists of two major components: the aircraft model and the expert model. The aircraft model consists of knowledge representing the physical characteristics and operating methodologies of the system. It employs a number of physical components of an aircraft. Each component assumes a small number of states. Briefly, the components range in complexity from a cockpit or fuel system to a switch or fuel gauge. The states of components include such things as switch positions and fuel tank levels.

The expert model consists of the fuel system categories and emergency conditions along with each procedure. This part of the program provides the following functionality: diagnosis of system emergencies, and development of solutions to these problem. The component that the expert model interacts with depends upon the current user's mode. If the system is in the reactive mode then the expert model interacts exclusively with the communication model. While in the instruction mode the expert model interacts with both the tutor and communication models. Information in the expert model was derived from the Naval Air Training and Operating Procedures Standardization Program (NATOPS) flight manual for F-4 operators and by interviewing known experts.

56

Declarative and procedural knowledge are both represented in the knowledge model. The model of the aircraft contains: (1) declarative knowledge about the aircraft description and about the state of the aircraft, and (2) procedural knowledge that changes the state of the aircraft. The expert model contains procedural knowledge required to reason and solve problems about the domain.

a. PEPTR Aircraft Model

The aircraft model is a representation of a physical aircraft. The state of the aircraft can be changed by manipulating the model. Reasoning about the state of a physical aircraft can be accomplished by using the model. The knowledge representation scheme used to represent the declarative and procedural knowledge in the aircraft model is a form of structured object representations called objects. The aircraft is designed as a composite object made up of different components. We represent components of an aircraft using objects. An object has slots containing attributes that describe the aircraft component and information about its relationship with other components. For example, an object called TEST-F4 has three slots, each containing pointers to other objects that represent three major components of the aircraft: cockpit, electrical system and fuel system. The cockpit object also contains three slots with pointers for values to objects that represent various panels in the cockpit: circuit breaker panel, electrical panel and fuel system panel. The fuel system panel object has slots that reference objects that represent all of the switches, gauges, and lights in the panel. The objects that represent the switches have a POSITION slot that contains a value that indicates the state of the switch.

We used a procedural representation scheme to represent constraint knowledge, a type of meta-knowledge. Constraint knowledge in the aircraft model includes the effect of an object on one or more other objects. For example, changing the amount of fuel in the tanks effects the rate of fuel change in the aircraft model. There-

57

fore, we attached active values to the slots that represent the fuel amount in the various tank objects. The active values ensured that the value of the fuel rate of change slot changed appropriately when the user modified the amount of fuel in the tanks. For each active value, we created a separate object with a method for updating the rate of fuel change in the aircraft. Since the model depicts an aircraft in a snapshot of time, the user cannot determine the rate of fuel flow in the same manner as in a real aircraft; that is, by comparing the amount of fuel between two gauges over time. Therefore, we display to the user a constant value that declares the fuel rate of change during a training session. There are certain rules and procedures that determine fuel rate of change just from the amount of fuel in the various fuel tanks. We have represented those procedures in active values. Therefore, whenever the user changes the fuel amounts in tanks, the active values are activated in order to update the fuel rate of change.

b. PEPTR Expert Model

The expert model captures the knowledge that a human expert uses to diagnose problems and determine solutions in an F4 aircraft. The knowledge representation scheme we used to represent this procedural and heuristic knowledge about the aircraft model was production rules.

(1) Production Rules. We chose production rules as the knowledge representation scheme for several reasons. Production rules provide modular representation of knowledge, skills and problem-solving methods. They encapsulate a piece of domain knowledge expertise. From a designers standpoint, this modularity reduces complexity by allowing incremental construction of the application, simplifies debugging efforts, and enhances extensions and modifications because rules can be added, deleted, or modified independently of each other. Production rules offer a uniform representation of knowledge to the designer or to another part of the system. The knowl-

edge is encoded in a rigid structure that is not imposed in a semantic net or procedural representation scheme. Production rules provide a natural form of representing knowledge. Human experts often explain their expertise as a matter of what to do in a certain situation (Barr and Feigenbaum, 1982, pp. 194-95). Additionally, production rules is a beneficial scheme because the information from the aircraft model needs to be combined opportunistically to achieve different diagnoses each time the expert model is invoked.

(2) Inference Engine. After choosing production rules to represent this knowledge, we needed to choose an inference engine. The term inference engine refers to the mechanism used to draw conclusions from the rules in the expert model and the data in the aircraft model. Since the KEE system provides forward and backward chaining as an inference engine for applications, we chose to use one for the reasoning process in PEPTR. First, we chose backward chaining to diagnose a problem because there are a small number of diagnoses that can be determined from a large set of facts. We originally chose forward chaining to determine the solution because there was one fact (the diagnosis) and a multitude of possible solutions. We used the KEE system facility KEEworlds to create new worlds to hypothetically reason about solutions and prevent affecting the original aircraft state. Upon implementation, we found that forward chaining had a major drawback. Rules were invoked in an exhaustive breadth-first search because the KEE system supports parallel reasoning and creation of new worlds. The combinatorics of this approach made it inefficient in practice, because hundreds of new worlds were created to reach all of the possible solutions. We realized that we needed the expert model to determine only one solution to a problem instead of finding all possible solutions. Therefore, we redesigned our solution rules and switched to backward chaining. This greatly reduced the number of new worlds created and increased the efficiency of the system.

59

(3) Diagnosis and Solution Rules. The user can invoke the expert model by mousing a button on the screen. This button is attached to a method of an object. This method queries the expert model for the problem. If the expert model identifies a problem, the method then queries the expert model for the solution to the problem. During this process, the method displays information to the user concerning the problem and the solution. The rules were written using the KEE system's TellAndAsk language. They are partitioned into several rule classes. Two major classes of rules are diagnosis rules and solution rules.

Diagnosis rules are used for diagnosing problems in the aircraft. Currently, this rule class contains rules for diagnosing fuel system problems. It is partitioned into rule subclasses that have different roles during the reasoning process. Figure 9 shows the classes and rule instances of diagnosis rules currently in our expert model.


```
                                                              FUEL.BOOST.FAILURE.NEW
                                                             /
                                    FUEL.PROBLEM.RULES  <----- FUEL.TRANS.PROB.NEW
                                                             \
                                                              OTHER.FUEL.PROB


                                                              CENTERLINE.TRANS.PROB
                                                             /
                                    FUEL.TRANS.PROB.RULES <--- EXT.WING.TRANS.PROB
                                                             \
                                                              INT.WING.TRANS.PROB

EMERGENCY.DIAGNOSIS.RULES

                                                              FUEL.LEAK.INT
                                                             /
                                    OTHER.FUEL.PROBLEMS.RULES <
                                                             \
                                                              REVERSE.TRANS.INT

                                    TOP.LEVEL.DIAGNOSIS.RULES ------ START
```
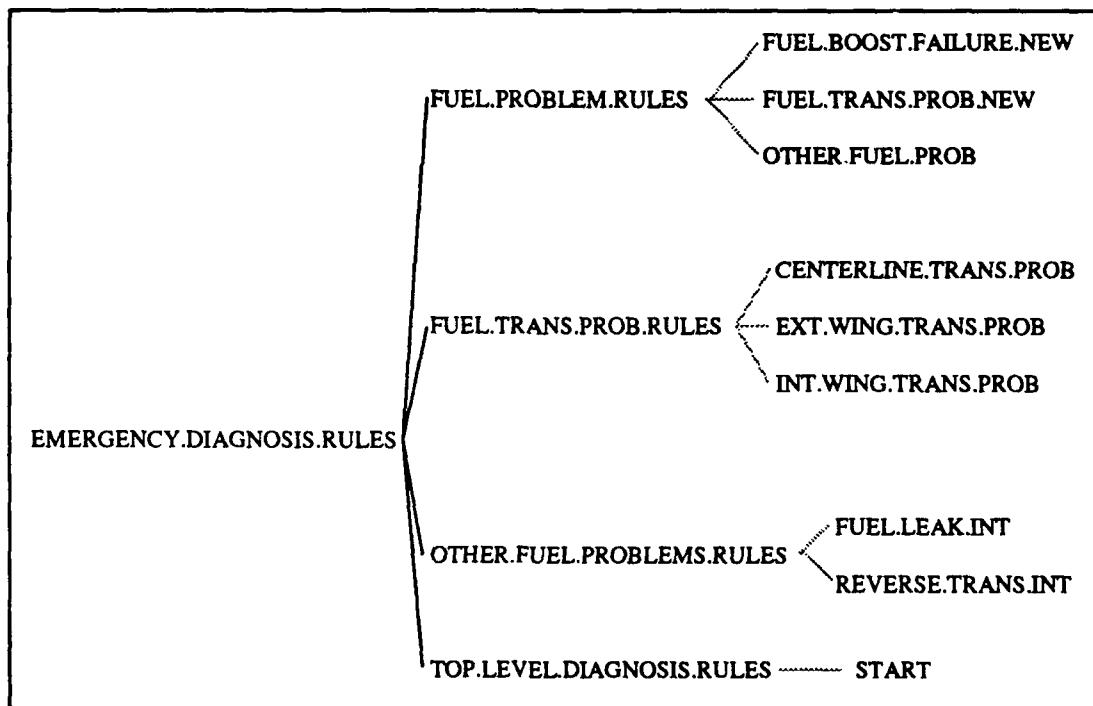
Figure 9 - Diagnosis Rules Hierarchy

The method explained earlier that invokes the expert model starts the backward chainer by querying the rule called START of the rule class TOP.LEVEL.DIAGNOSIS.RULE. This rule queries the rules in the rule classes that represent the major types of emergency problems such as fuel, electrical, fire, and hydraulic. Since the expert model currently contains only knowledge of fuel system problems, the START rule backward chains using only the rules in the rule class called FUEL.PROBLEM.RULES. This class of rules are intermediate rules created to reduce the difficulty in keeping track of everything necessary for each rule. Intermediate rules represent generalizations about a group of facts, thereby reducing redundancy and simplifying rules (Rowe, 1988, p.130). These rules each represent a specific type of fuel emergency problems. For example, one type of fuel problem occurs when their is a a fuel transfer problem in the fuel system. Another type exists when a fuel boost pump fails. Hence, these rules try to identify the type of fuel problem. They do so by doing further backward chaining. For example, the intermediate rule called FUEL.TRANS.PROB.NEW queries all of the rules in the rule class FUEL.TRANS.PROB.RULES. These are the primitive rules which determine the specific problem, if one exists. These rules contain multiple premises of two types. One type of premise checks the state of the aircraft, the other type checks facts asserted in the knowledge base. Checking the state of the aircraft involves a premise of a diagnosis rule looking at certain slot values of those objects in the aircraft model that can cause the problem corresponding to the conclusion of that diagnosis rule. Sometimes the state of an aircraft component applies to multiple problems. In this case, the conditions for all the applicable problems must be checked. This leads to the second type of premise. Checking facts in the knowledge base involves invoking backward chaining on a different class of rules from within backward chaining. Thus, a premise of this type invokes backward chaining on the rule class that contains the rule for determin-

61

ing one of the applicable problems and checks the knowledge base to see if that problem is asserted in the knowledge base. If all the premises of a diagnosis rule are satisfied then the diagnosed problem is asserted into the knowledge base. Figure 10 shows examples of a diagnosis rules from each rule class.

The solution rules are used to determine the solution to the problem if one exists. They are also partitioned into several smaller rule classes for the same reasons that the diagnosis rules are partitioned. The same method that invokes the backward chainer to diagnose a problem also invokes the backward chainer on a specific rule class of the solution rules. The process of determining the solution is conceptually the same as for diagnosing the problem. Intermediate rules are used in the process. Another class of rules determines the operations which the user must implement to change the state of the aircraft and asserts this as a fact into the knowledge base.

START Rule

```
((IF (OR (FIND (TEXT (THE FLIGHT.CONTROL.PROBLEM IS ?PROB)))
         (FIND (TEXT (THE HYDRAULIC.PROBLEM IS ?PROB)))
         (FIND (TEXT (THE ELECTRICAL.PROBLEM IS ?PROB)))
         (FIND (TEXT (THE FIRE.PROBLEM IS ?PROB)))
         (FIND (TEXT (THE FUEL PROBLEM IS ?PROB)) USING FUEL.PROBLEM.RULES))
  THEN
     (TEXT (THE SYSTEM PROBLEM IS ?PROB))))
```

FUEL.TRANS.PROB.NEW Rule

```
((IF (FIND (TEXT (THE FUEL TRANSFER PROBLEM IS ?FUEL.PROB))
           USING FUEL.TRANS.PROB.RULES)
  THEN
     (TEXT (THE FUEL PROBLEM IS ?FUEL.PROB))))
```

INT.WING.TRANS.PROB Rule

```
((IF (LISP (> (THE FUEL OF TEST-FUEL.QUAN.IND.CNTR)
              (THE FUEL OF TEST-FUEL.QUAN.IND.TAPE)))
     (FIND (EQUAL (THE RATE.OF.CHANGE OF TEST-FUEL.QUAN.IND.CNTR)
                  (THE RATE.OF.CHANGE OF TEST-FUEL.QUAN.IND.TAPE)) USING NO.RULES)
          (LISP (< (THE FUEL OF TEST-FUEL.QUAN.IND.TAPE)
                   (THE MAXFUEL OF TEST-FUEL.QUAN.IND.TAPE)))
          (CANT.FIND
             (FIND (TEXT
                (THE FUEL TRANSFER PROBLEM IS CENTERLINE.TRANSFER.PROBLEM))
                USING FUEL.TRANS.PROB.RULES))
          (CANT.FIND
             (FIND (TEXT
                (THE FUEL TRANSFER PROBLEM IS EXT.TRANSFER.PROBLEM))
                USING FUEL.TRANS.PROB.RULES))
  THEN
     (TEXT (THE FUEL TRANSFER PROBLEM IS INT.WING.TRANSFER.PROBLEM))))
```

Figure 10 - Examples of Diagnosis Rules

2. PEPTR Tutor Model

The tutor model manages the training session through close coordination with the student, domain knowledge and communication models. At the beginning of a session, the tutor interacts with the student model to determine the in-flight emer-

63

gency which should be presented to the student. If the student is known to be deficient at solving the problem, then the tutor model interacts with the domain knowledge model to present instruction on the in-flight emergency. When all instruction is completed, the tutor model interacts with the domain knowledge model to present an in-flight emergency for the student to solve. When the problem is introduced, the tutor invokes the expert model to determine the correct solution. Through the course of the problem solving session, the tutor model accumulates student responses. It then gives the student model the solution and student responses for the student model to compare and update the student's knowledge state appropriately. Finally, the tutor model interacts with the domain knowledge and communication models to provide feedback to the student.

The tutor model of PEPTR contains procedural knowledge associated with a tutor. We used a procedural representation scheme to represent this procedural knowledge. To implement the tutor model, we created an object which contains one method called TEACH.F4 that represents the procedural tutoring process. The TEACH.F4 method consists of two main procedures.

The first procedure is outlined as follows. First, the method activates a rule class called STUDENT.DEFICIENCY.RULES in the student model to find an emergency condition which the student is deficient in. If one exists, the tutor model tells the domain knowledge model to present the instruction about the emergency condition to the student. After the instruction is completed, the TEACH.F4 method tells the student model to upgrade the student's knowledge state to proficient. The TEACH.F4 method repeats this procedure until no deficient emergency conditions exist in the student model.

The second procedure is then implemented. The TEACH.F4 method activates another rule class called PRESENT.EMERGENCY.RULES in the student model to

find an emergency condition which the student is proficient in. Next, the method activates an object in the domain knowledge model corresponding to the emergency condition obtained from the student model to initialize the aircraft model to a state that causes the emergency condition. The TEACH.F4 method then invokes the expert model for the solution and temporarily stores the information in a slot. The method instructs the student to diagnose the current problem and implement a solution to solve the problem. This is standard information in the form of text, stored in a slot of the tutor object. Next, the TEACH.F4 method tells the communications model to display the aircraft interface so the student can look at the switches and gauges in order to diagnose the problem. Upon diagnosing the problem, the student changes switch positions to solve the problem. The tutor method records all of these changes. When the student concludes all changes, the TEACH.F4 method sends the system and student solutions to the student model to compare the students solution to the expert model's solution. This procedure is repeated until the student is either deficient in an emergency condition or is proficient in all emergency conditions.

3. PEPTR Student Model

The student model is used to maintain a record of the student's performance. It represents the student's proficiency at diagnosing each emergency condition and applying the correct procedure to solve the problem. The student model interacts with the domain knowledge model to represent the student's knowledge state.

The type of knowledge that the student model currently represents is declarative knowledge. The declarative knowledge is the part of the student model that provides a reflection of the student's knowledge state during each tutoring session. We used the production rules in the knowledge model to preserve the student's knowledge state. Since rules are represented as objects in the KEE system, each rule object has the same characteristics as the basic object in KEE. Therefore, rule objects

65

can contain slots. We created a slot in each production rule that encapsulates a particular concept that the student must know. This slot contains a value of deficient or proficient, which represents the student's knowledge of the concept in that rule. During a tutoring session, the student model activates applicable rules to update the value of this slot to reflect the student's current proficiency.

4. PEPTR Communication Model

This is the interface. Its purpose is threefold: to graphically represent the cockpit of an F-4 aircraft to the pilot, to provide windows for student input and system feedback, and to furnish a mechanism for the student to invoke the expert system while in the reaction mode.

a. Interface

The interface contains panels of switches and gauges that depict the current switch positions and state of the aircraft. The pilot can use the mouse to change switch positions and the state of the aircraft (e.g., fuel capacity, fuel rate of flow) to either initiate an emergency condition or to apply a solution. Windows appear when administrative information is required and during a problem solving session for feedback. Figure 11 illustrates a typical screen during a tutoring session.

The interface is designed for use in two modes of operation: reaction mode and instruction mode. During the reaction mode, the student can freely change the state of the aircraft to learn what in-flight emergencies are caused by various aircraft states and to learn which changes lead to a solution. During this mode, the interface provides a button so the student can invoke the expert model to determine the diagnosis and correct solution. During the instruction mode, the interface is displayed to the student at a certain time during the tutoring process. It is displayed in a predetermined state dependent upon the in-flight emergency selected for the student to solve. In this mode, the student makes only the changes required to solve the problem.

66
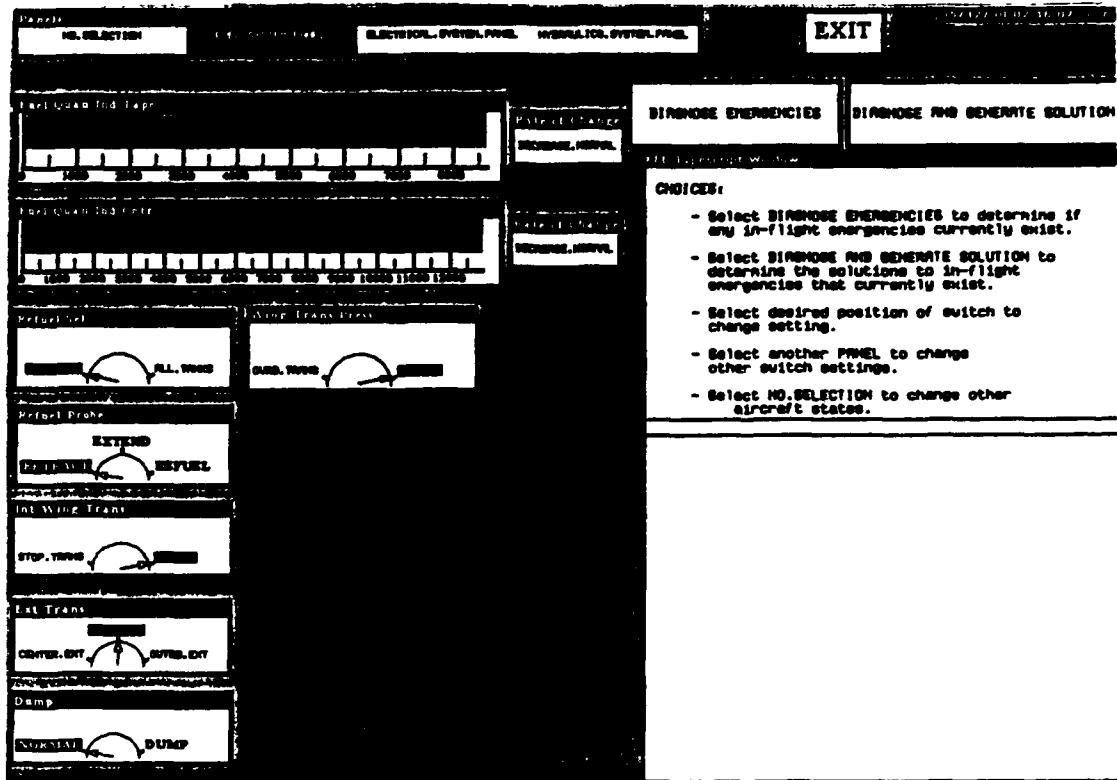
Figure 11 - PEPTR Interface

b. KEE Image and Graphics Facilities

The communication model for PEPTR is essential to the tutoring system. It not only acts as the user interface, but graphically represents a cockpit of an aircraft and translates user input into a format that the domain knowledge model can use. The type of knowledge in the communication model is procedural. We used a procedural representation scheme to represent this knowledge.

We used the ActiveImages facility that the KEE system provides to graphically represent components of a cockpit. Images of these components are attached to slots of objects in the aircraft model. We employed seven types of ActiveImages:

67

state dials, horizontal push buttons, vertical push buttons, method actuators, horizontal bar graphs, horizontal bar graph actuators, and simple value displays.

State dials display all possible values of a slot around the top of a dial. The needle points to the current value. The user can left-click on the image to change the value. We represented switches and circuit breakers in the cockpit with state dials.

Horizontal and vertical push buttons display all possible values of the attached slot with the current value highlighted. These are actuator images that enable the user to change the slot's value. To select a slot value, the user left-clicks on the desired value. All other values are deselected. We used a horizontal push button for the user to select a panel in the cockpit. Since a computer screen is not large enough to display all of the switches, buttons, and gauges of a cockpit, we designed the interface to display a selected part of the cockpit as determined by the user. Selection of a panel displays all of the images that represent switches and gauges corresponding to the type of panel (i.e., the electrical panel contains images of circuit breakers). We used a vertical push button for the user to select the fuel rate in the aircraft.

Method actuators provide a way to activate methods. If the user left-clicks on a method actuator, the method in the attached slot starts running. While the method is running, the method actuator image remains inverted. When the method has finished running, the image returns to its normal state. Text can be inserted in the method actuator image that describes or labels the operation it performs. We used method actuators for the user to start a method that invokes the expert model to diagnose a problem and determine a solution.

Horizontal bar graphs show the value of a slot via a variable length horizontal bar. The axis is underneath the bar. Horizontal bar graph actuators are similar, but also allow the user to change the slot value by left-clicking a new length for the bar. We used horizontal bar graphs to display the fuel levels of various fuel tanks

on the aircraft. We used horizontal bar graph actuators for the user to change the fuel level in the fuel tanks.

Simple value displays can be attached to any slot and displays the value as a string of text or numbers. It is intended to display a slot with just one value; however, it can be shaped to display all values of a slot. We used simple value displays for the user to see the rate of fuel change.

All images contain functionality derived from knowledge bases within the KEE system. This functionality is in the form of methods range from the actions which are performed when a mouse button is clicked to the action performed when a slot value is changed. These methods represent the procedural knowledge required to implement the user's input.

E. EXTENSIONS

A useful extension to the domain knowledge model of PEPTR is a set of emergency situations for each in-flight emergency. The emergency situations of an aircraft would be objects with slots that contain declarative and procedural knowledge to manipulate the state of the aircraft model. One slot would contain only those aircraft components and their state values that cause the certain emergency condition. Another slot would contain a method for introducing an in-flight emergency by changing the state values of components in the aircraft model. This would initialize the aircraft model for a problem solving session on a particular emergency condition.

One extension necessary for the tutoring process is the addition of an instruction model. This part of the domain knowledge model would contain displays of text and graphics for teaching the student about the emergency conditions of an aircraft. A group of objects corresponding to each of the emergency conditions would have slots

69

that contain the text to present to the student for instruction and a list of applicable switches, gauges or other aircraft components to display to the student.

The following extension to the tutor model would make the tutor model more intelligent and improve the feedback to the student. Currently, the teach method reflects somewhat the "discovery" method of learning. It just records the student's solution without any intervention, giving the student full initiative to make any changes desired. A better teaching strategy would be to implement the coaching method during the student's problem solving process. After each student operation, the teach method would invoke the expert model to replan. The method would then invoke a set of rules that would check if the operation was correct or incorrect and provide feedback accordingly. An incorrect response would result if the student operation caused three things. First, the operation caused additional emergency problems. Second, the operation resulted in no change; the problem is not reduced. Third, the operation prevented realization of any solution to the problem. The teach method would request from the domain knowledge model appropriate feedback depending upon the result of the student operation.

The student model needs additional development to be able to assess a student's knowledge state. We would use production rules and a procedural representation scheme to represent the procedural knowledge in the student model. Production rules would be written for the following purposes. One set of rules called STU-DENT.DEFICIENCY.RULES, would determine the emergency condition which the student is deficient at diagnosing or solving. Another set of rules called PRESENT.EMERGENCY.RULES, would randomly determine an emergency condition for instruction if the student was proficient in all emergency conditions. We would use a procedural representation scheme to represent the procedural knowledge in the student model. The procedure would compare the expert model's solution to a

70

problem with the student's solution. The method in the student model would identify three things: the changes which the student made that were not in the expert model's solution, the changes in the expert model's solution which the student failed to implement, and the correct changes that the student made. For each change that the student made incorrectly, failed to make, or made correctly, the student model method would identify and activate the rule in the expert model which represents that particular expertise to update the student knowledge state appropriately.

VIII. CONCLUSION

A. RESEARCH LESSONS

Most ICAI systems consist of the four general components required for effective instruction. Each component captures a different type of knowledge: knowledge about the domain, knowledge about teaching, knowledge about the student, and knowledge about communication.

In general, we believe there are three forms of knowledge: declarative knowledge, procedural knowledge, and meta knowledge.

AI researchers and others have developed numerous knowledge representation schemes over the past 30 years. Most of these schemes fall into five general categories. Each category of knowledge representation schemes offers both advantages and disadvantages to the system designer.

The most important aspect of this research is the realization that designers of ICAI systems need to employ various knowledge representation schemes to represent the different forms of knowledge in an ICAI system. This is evident by studying knowledge and knowledge representation schemes, and by studying and implementing ICAI systems.

B. MAPTR AND PEPTR

The ICAI systems we developed are both diverse and similar in many respects. The knowledge domains are quite different as well as the interfaces for each application. The general components of a typical ICAI system were used as a framework for both systems.

72

MAPTR is a frame-based system that generates a lesson plan tailored to the student and provides instruction and evaluation through displays of text and graphics. A taxonomy of tasks represents the classification of knowledge into chunks and the relationships among the chunks of knowledge. The tutor model extensively interacts with the knowledge and student models to determine the tasks to be taught, the sequence of the instruction, and the training mode.

PEPTR is a reasoning application for diagnosing aircraft emergency conditions and determining solutions to remedy problems. A frame-based aircraft model is utilized for representing various aircraft components and their defined states. Reasoning about the model is accomplished using a production system. An extensive interface resembling the cockpit of an aircraft is provided for the pilot to define or change the state of the aircraft. Currently, a pilot retains full initiative during the learning process. Future extensions to PEPTR include providing an instructional mode whereby the system controls the tutoring process. In addition, the student model will be extended to include features which allows the system to capture and assess a student's knowledge state. The knowledge model will also be extended to initialize the state of the aircraft for a specific emergency condition and to provide instruction to the student.

C. ACCOMPLISHMENTS

We designed two distinctly different ICAI systems in an environment that supported multiple, integrated knowledge representation schemes. The most important issue in developing both tutoring applications was choosing the appropriate way of representing the knowledge in each component of the tutoring systems. To accomplish this, we first determined the types of knowledge to be represented in each component. We then selected an appropriate representation scheme for each knowledge

73

type compatible with the tools and techniques provided by the KEE software development environment.

Both applications contain all three forms of knowledge. We used four different knowledge representation schemes to represent knowledge in the four components of each application. The KEE system supported the creation of objects to represent declarative knowledge and procedural knowledge. It also supported the creation of a semantic network to represent relational knowledge. In addition, the KEE system supported the creation of production systems to represent procedural knowledge and meta knowledge.

Clearly, an ICAI system designer is faced with the complex task of determining the most natural and applicable knowledge representation scheme for the various forms of knowledge contained in the different components of an ICAI system. Thus, to reduce the complexity of this task, the designer develop ICAI systems in a programming environment that supports multiple, integrated knowledge representation schemes. With such a programming environment, designers can implement the most natural knowledge representation schemes for all of the knowledge in the system.

D. FUTURE WORK

Both applications need additional development to make them become suitable ICAI systems for assisting trainers in the instruction of their respective domain knowledge. The domain knowledge model of both systems could be expanded to include the complete body of knowledge required for the users to be proficient at their respective tasks. Specifically, PEPTR needs two extensions in the domain knowledge model. One, a set of emergency situations could be added to initialize the aircraft model for a problem solving session. Two, an instruction model should be incorporated to teach the student about the emergency conditions of an aircraft. The

tutor models of both systems provide limited teaching strategies. Additional tutoring strategies could be incorporated to serve the needs of the various users of the systems. The student model in PEPTR needs additional development to be able to assess the student's knowledge state. With these improvements, both ICAI systems could become useful tools for teaching map reading and emergency aircraft procedures.

APPENDIX A - SOURCE CODE

All source code for both ICAI systems is located on the Texas Instrument Explorer server called "EXP3". The KEE package for running the applications is located on Texas Instrument Explorer called "EXP1". After loading the KEE package, use the following pathnames to load either application.

- MAPTR - exp3:studentthesis.maptr;maptrdesk.lisp

- PEPTR - exp3:studentthesis.peptr;peptrdesk.lisp

Each application contains several files of source code. Some files contain code generated by KEE for different knowledge bases in an application. A knowledge base is a feature of KEE's frame system that allows a designer to group related objects of an application. Other files contain user defined Lisp functions which are methods of various objects in the knowledge bases. Each application consists of three knowledge bases. The files for the three knowledge bases in MAPTR are named chaparral.u, system.u, and systemtasks.u. Three additional files in MAPTR contain LISP functions for objects of each knowledge base: chaparralfn.lisp, systemfn.lisp, and systemtasksfn.lisp. The files for the three knowledge bases in PEPTR are named f4.u, rules.u, and system.u. The file f4.u contains the objects that represent the aircraft model. The rules.u file contains the rule objects for the expert model. The Lisp function files are called f4fn.lisp, rulesfn.lisp, and systemfn.lisp.

APPENDIX B - MAPTR SAMPLE TUTORING SESSION

The following screen images depict various views of the interface to MAPTR during a typical tutoring session. Figure 12 illustrates a general screen image with three windows. The lower left window is provided for the user to input textual information into the program and for the program to display messages to the user. The upper left window displays graphical pictures while the upper right window displays text of domain knowledge. A menu is provided in the lower right corner for the user to either exit the program or continue the instruction. Figures 12, 13, and 14 illustrate typical graphics and text which the program displays for instruction. Figure 15 illustrates an evaluation of a task. A menu is provided in the lower right corner for the student to select a response.
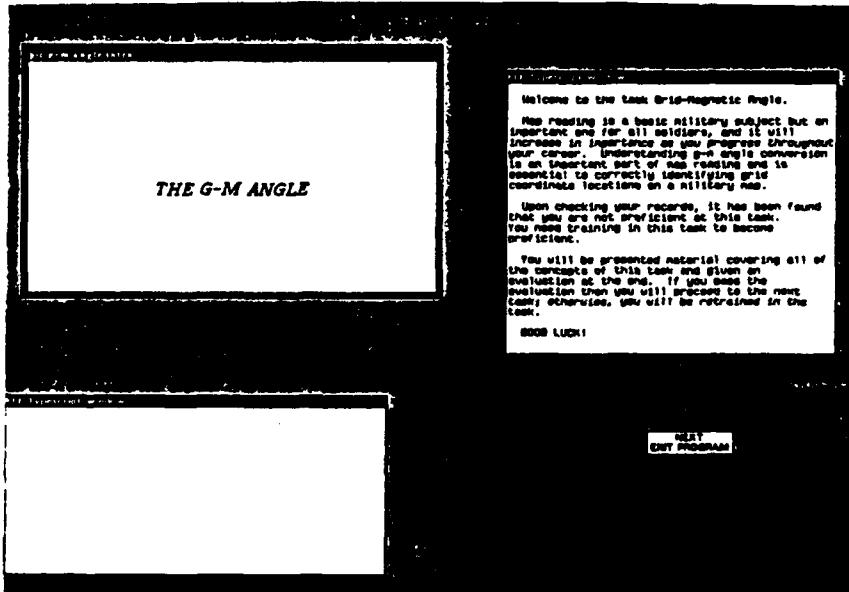
Figure 12 - MAPTR Window Layout
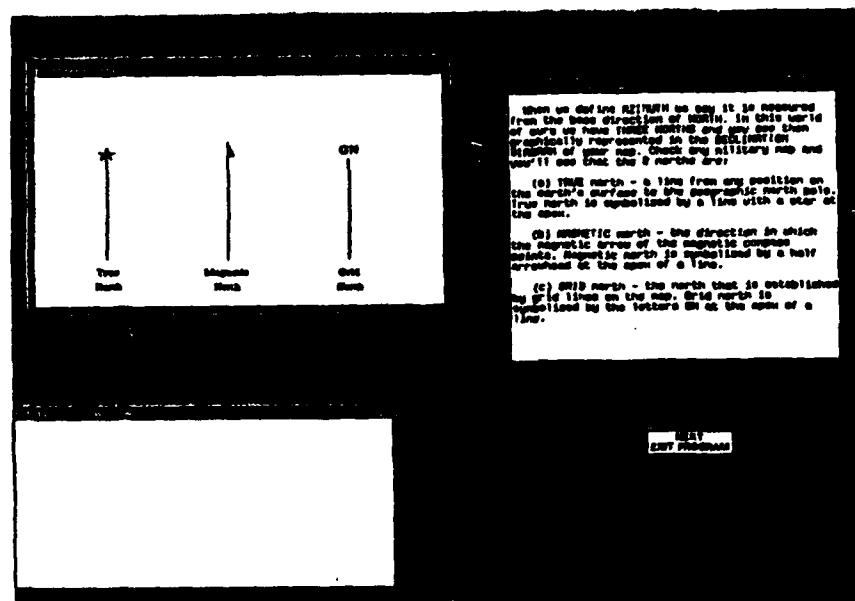
Figure 13 - MAPTR Instruction Display
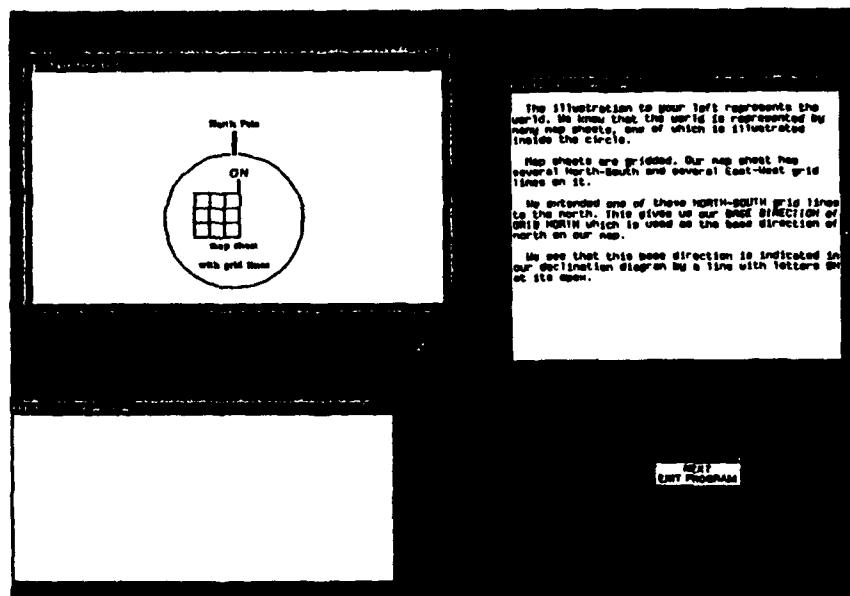
Figure 14 - MAPTR Instruction Display
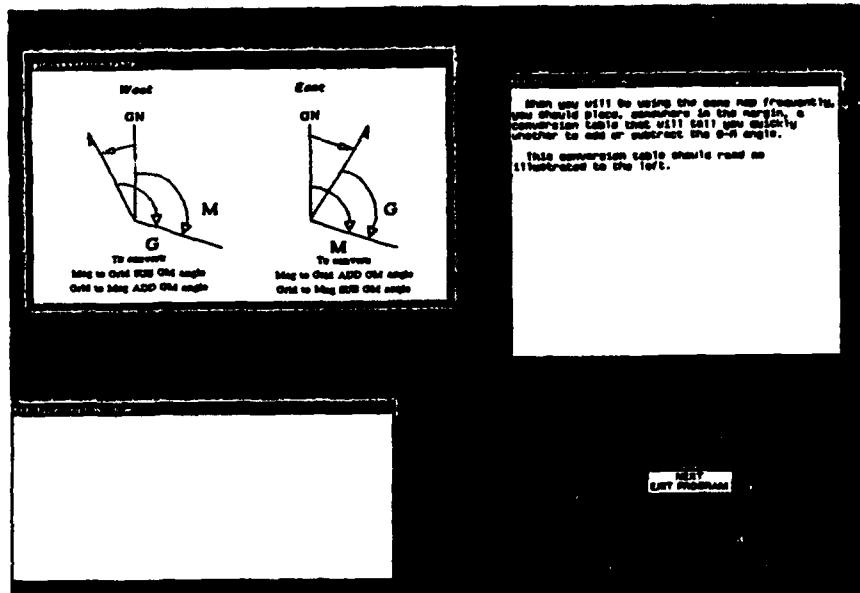
Figure 15 - MAPTR Evaluation Display

APPENDIX C - PEPTR SAMPLE TUTORING SESSION

The following screen images depict various views of the interface to PEPTR during a typical tutoring session. Descriptions of the screen images are also provided.

Figure 16 shows the opening screen of PEPTR. The student clicks in the login button with the mouse to begin the program.
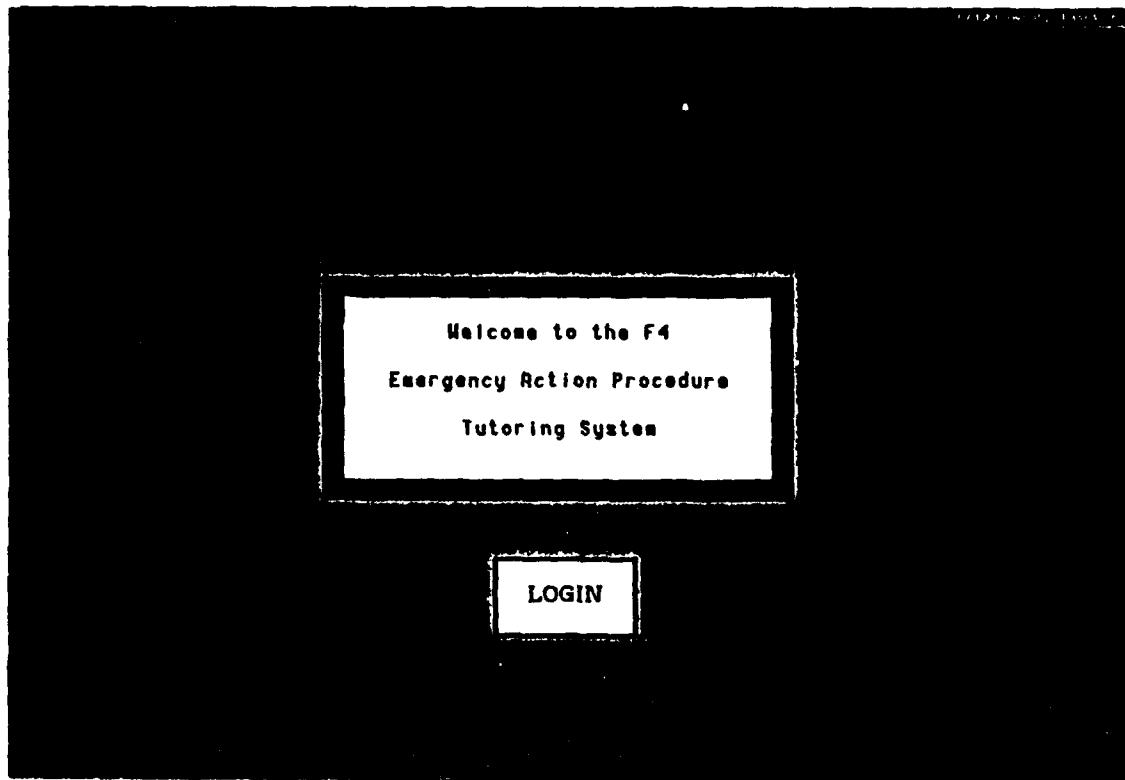
Figure 16 - PEPTR Login Display

Figures 17 and 18 show how a student enters administrative data and the menu the student uses to select the mode of instruction (tutor or reactive).
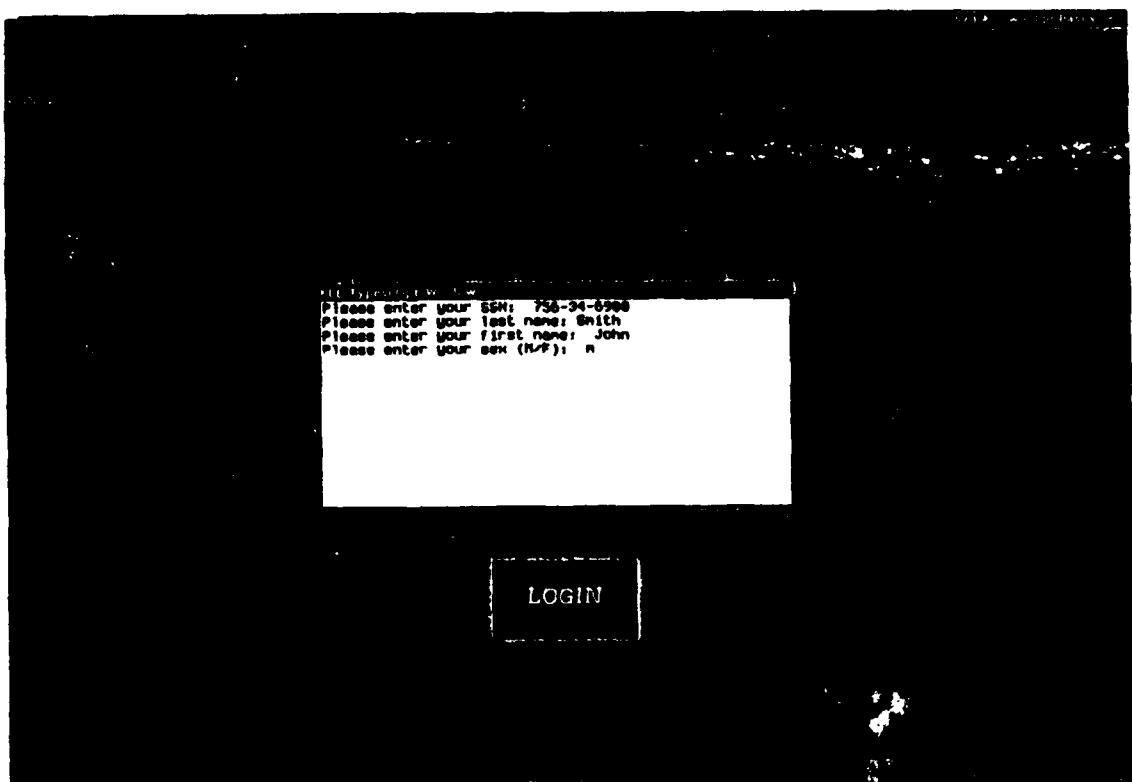
Figure 17 - PEPTR Administrative Display
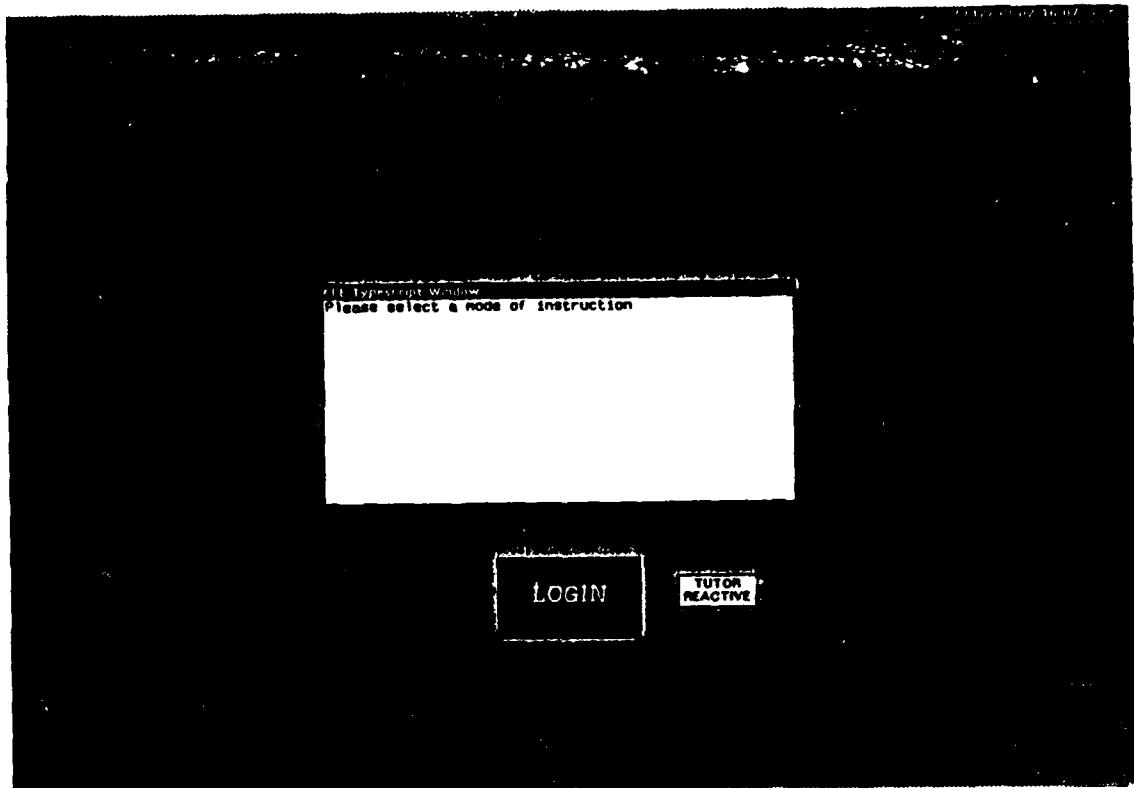
Figure 18 - PEPTR Instruction Mode Display

Figure 19 illustrates the general screen display which the student uses to select different actions to include displaying aircraft panels, changing the aircraft state, and invoking the expert model to diagnose a problem and show the solution. The student displays a panel by clicking on the top image labeled "Panels". The student can display additional screen images for changing the state of the aircraft by clicking on the button labeled "SET AIRCRAFT STATE". The button labeled "DIAGNOSE EMERGENCIES" invokes the expert model to determine the emergency condition if one exists. The button labeled "DIAGNOSE AND GENERATE SOLUTION" also diagnoses the problem and additionally provides the solution to correct the problem. The exit button allows the student to halt the program.
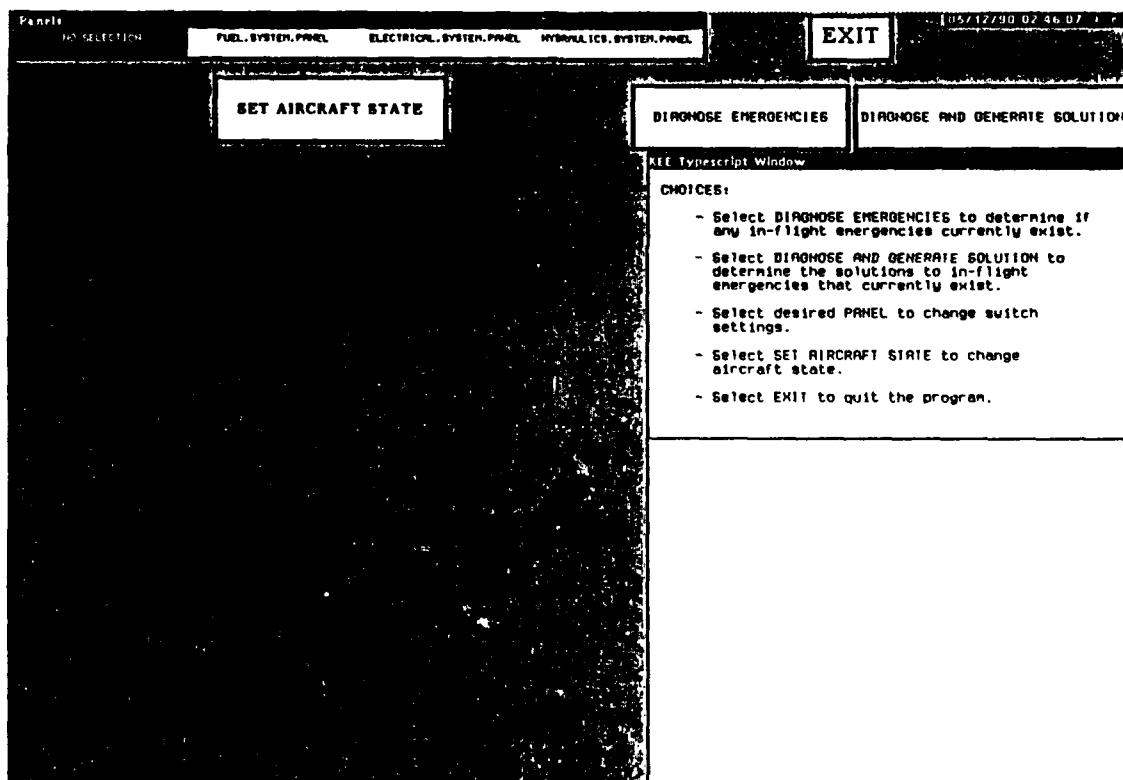
Figure 19 - PEPTR General Display

Figure 20 shows the message which the program displays if the student clicks on the "DIAGNOSE EMERGENCIES" button before changing the aircraft state. The aircraft model is initialized to a normal state.
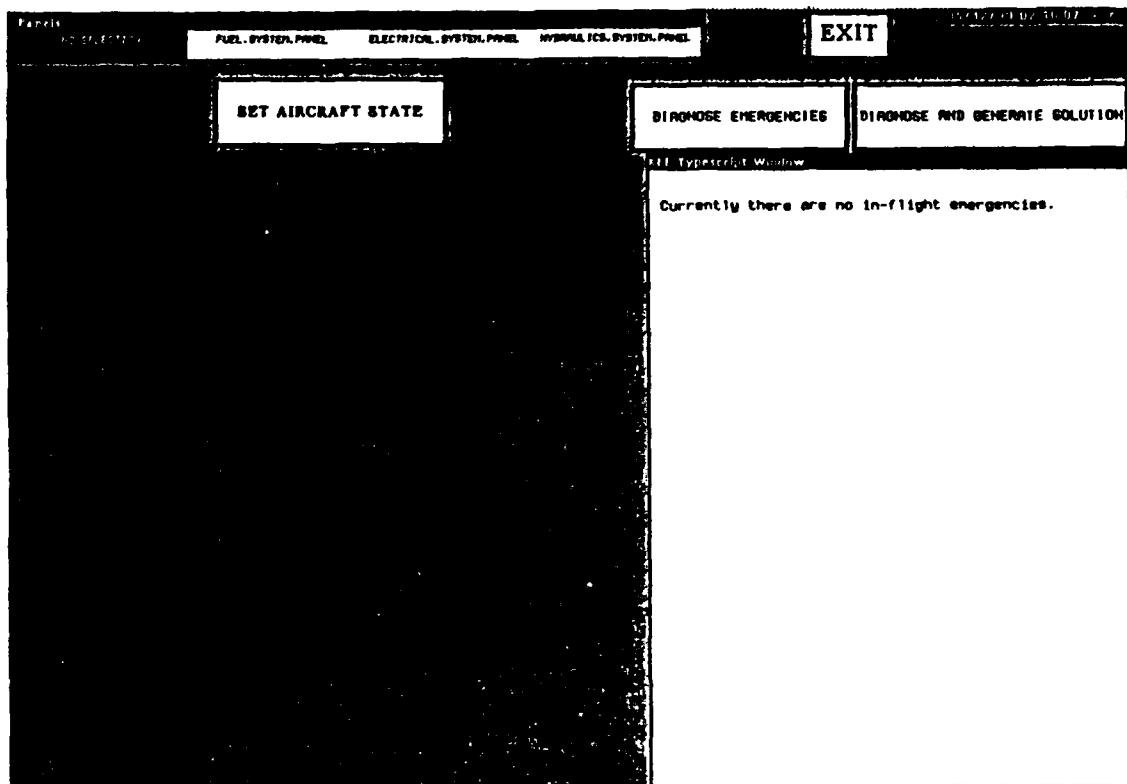
Figure 20 - PEPTR Diagnose Emergencies Display

Figure 21 shows the various switches and fuel gauges that are contained in the fuel system panel. They are displayed when the student clicks on the image labeled "Panels". Figure 22 shows the circuit breakers contained in the electrical system panel. Note, two of the circuit breakers are "out". Instructions are provided in the KEE typescript window.
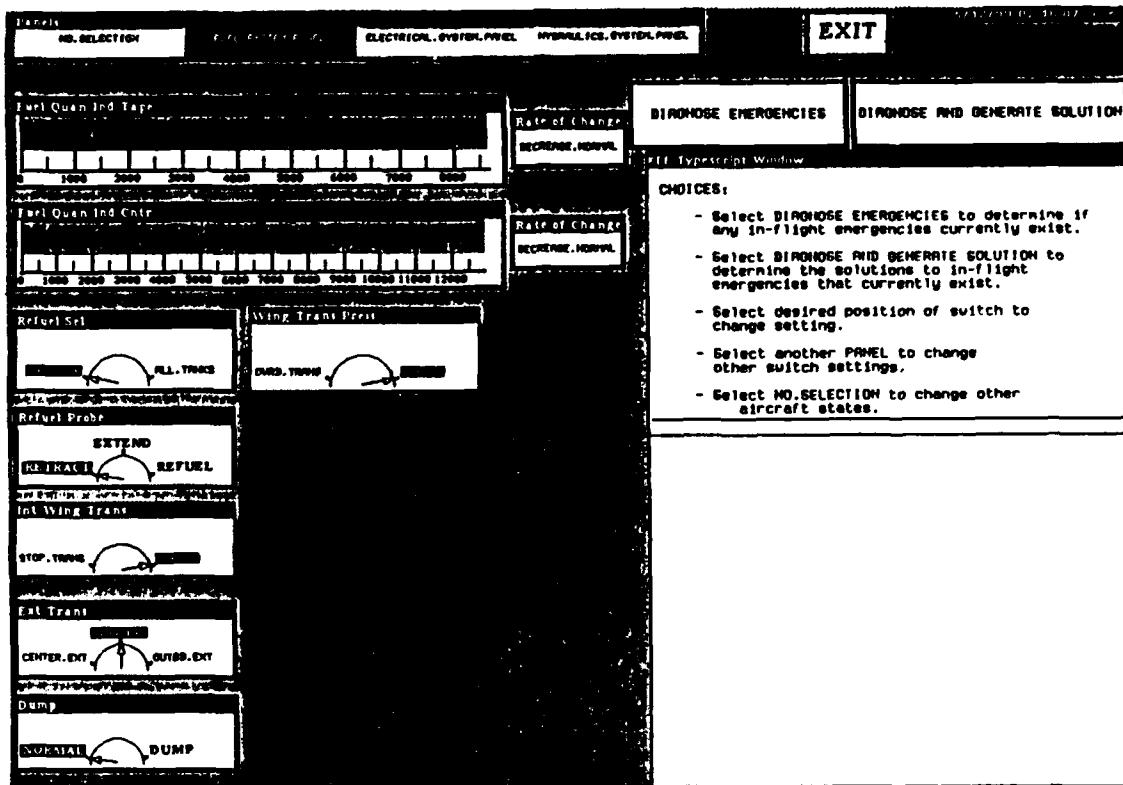
Figure 21 - PEPTR Fuel System Panel Display

85
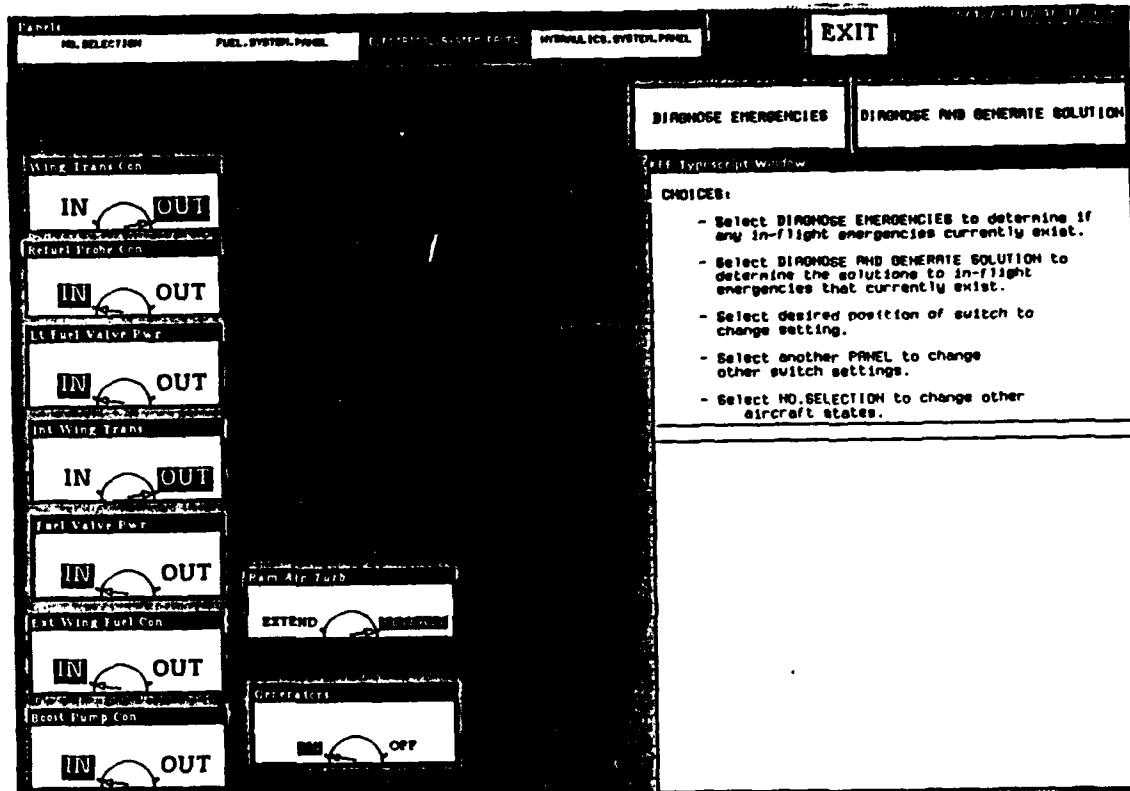
Figure 22 - PEPTR Electrical System Panel Display

Figures 23 and 24 show the images that are displayed when the student selects the "SET AIRCRAFT STATE" button. The student can click on the fuel gauge actuators in figure 24 to set the quantity of fuel desired in each fuel tank. Note, the fuel in the fuselage tank has been reduced to 6000 lbs. The button labeled "ADDITIONAL.STATES" displays the images in figure 24. The student can select the desired rate of fuel flow by clicking on one of the values in the "change rate" images.
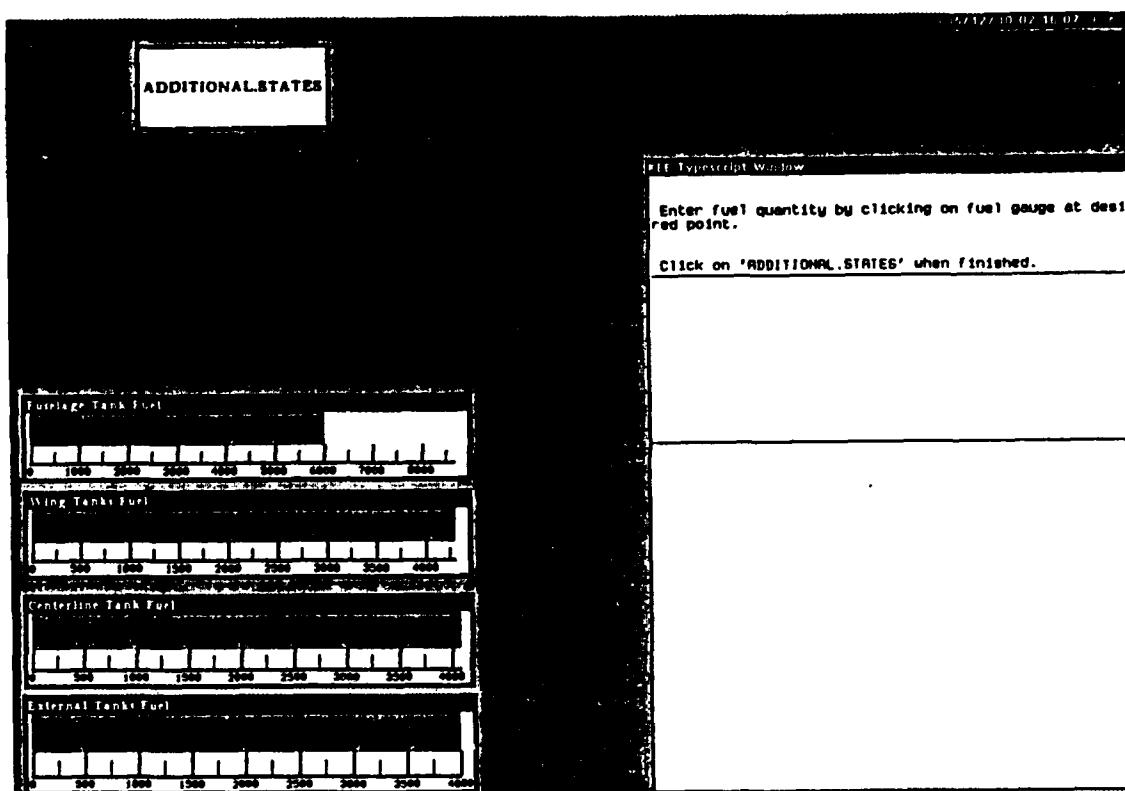
Figure 23 - PEPTR Set Aircraft State Display
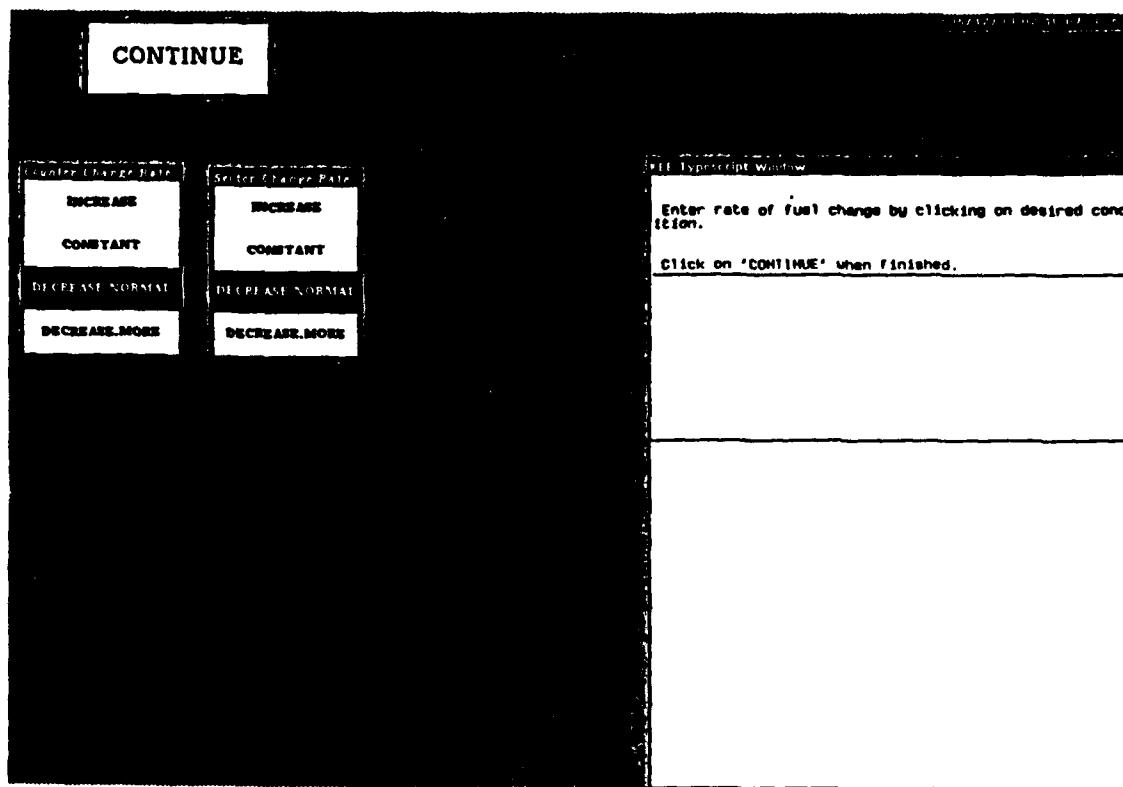
Figure 24 - PEPTR Additional States Display

Figure 25 shows the message which is displayed after the student has clicked on the "DIAGNOSE EMERGENCIES" button to invoke the expert model. The two circuit breakers that were "out" in figure 22 combined with the reduction of fuel shown in figure 23 produced the emergency condition shown in the message. Figure 26 shows the solution which the program displays if the "DIAGNOSE AND GENERATE SOLUTION" button is actuated.
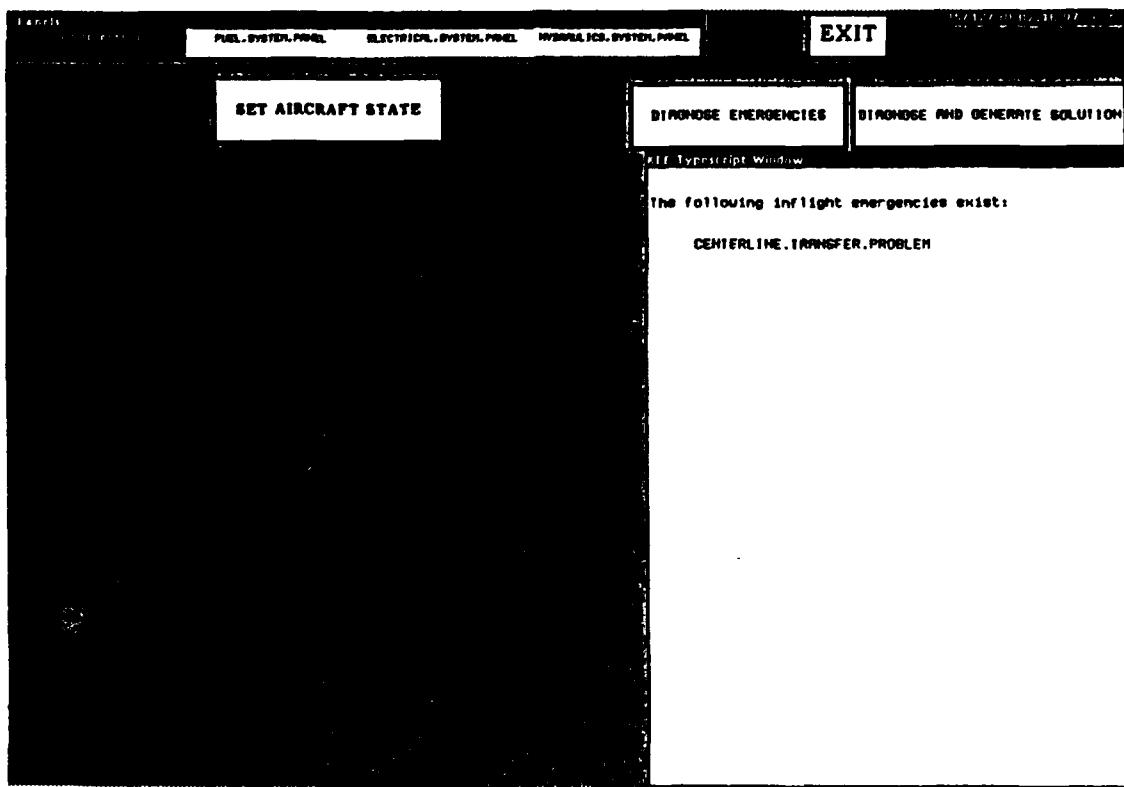

Figure 25 - PEPTR Diagnose Emergencies Display
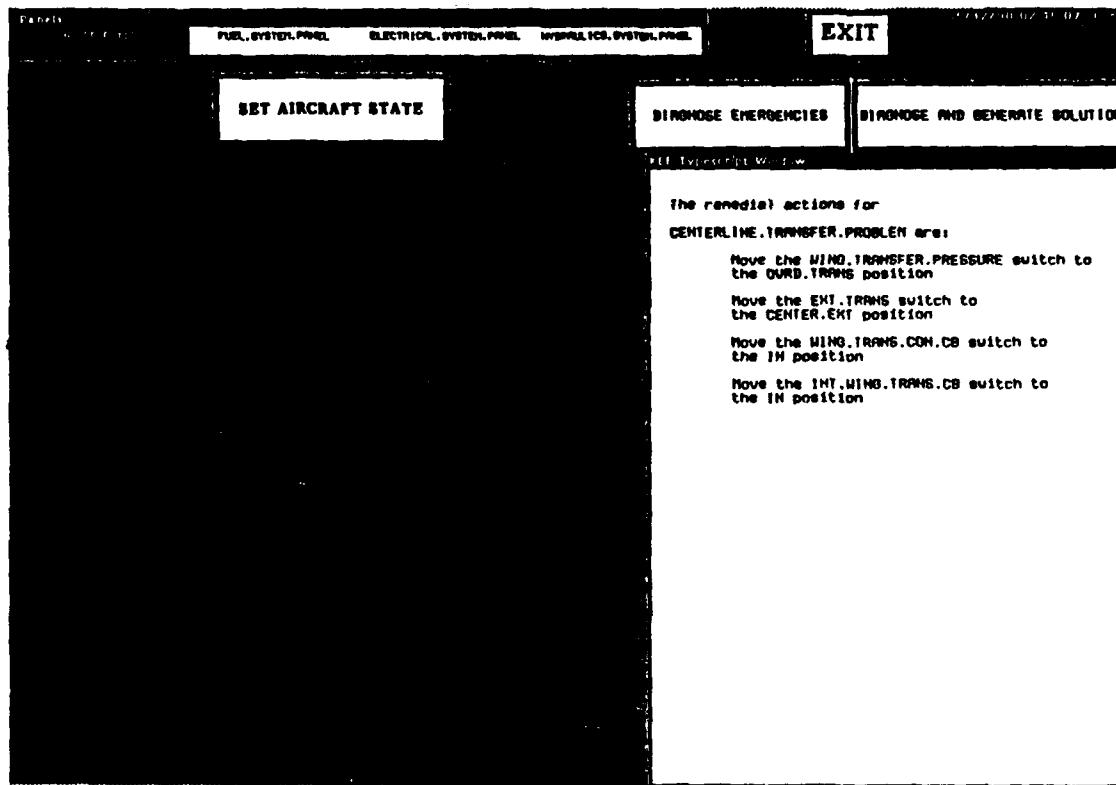
Figure 26 - PEPTR Diagnose And Generate Solution Display

Figure 27 shows the state of the circuit breakers after the student initiated the corresponding remedial actions concerning the diagnosed problem. Figure 28 shows the message which is displayed after the student has clicked on the "DIAGNOSE EMERGENCIES" button again to invoke the expert model. Since the student initiated only a partial solution, the expert model diagnosed the same problem and displayed the remaining remedial actions to correct the problem. Figure 29 shows the state of the switches on the fuel system panel after the student initiated the remaining remedial actions which alleviated the emergency condition.
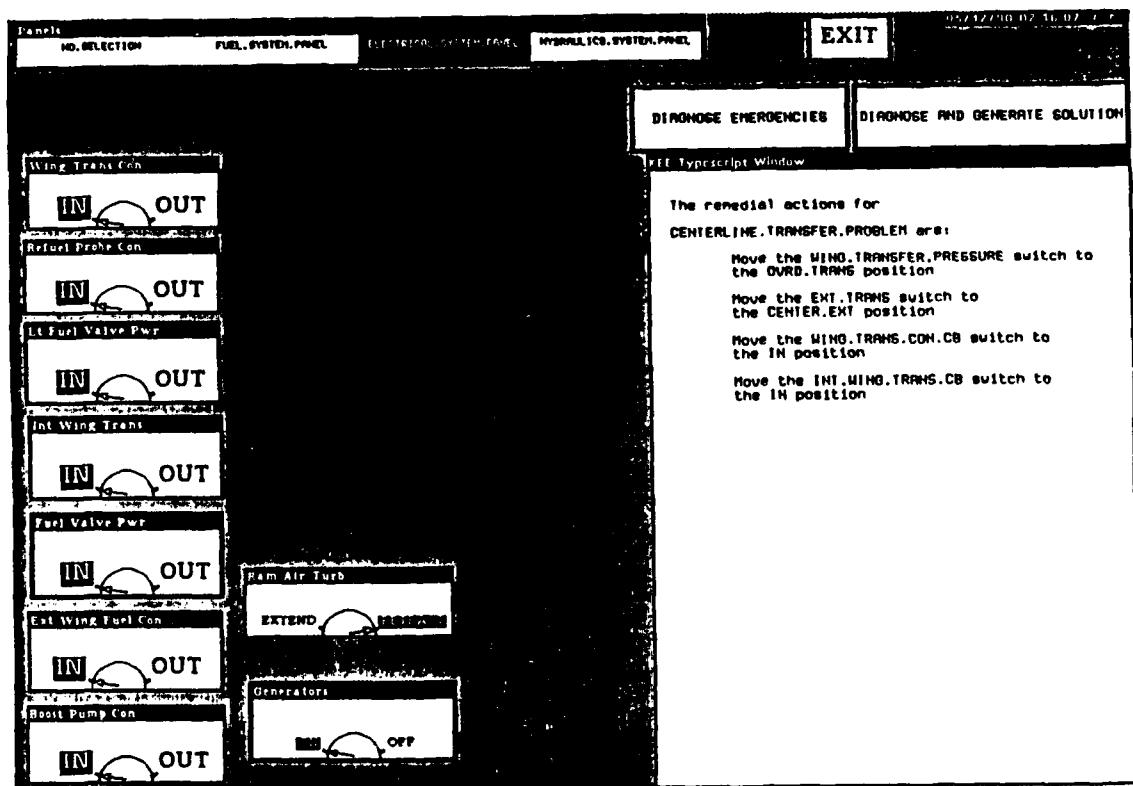

**Figure 27 - PEPTR Changing Electrical System State Display**
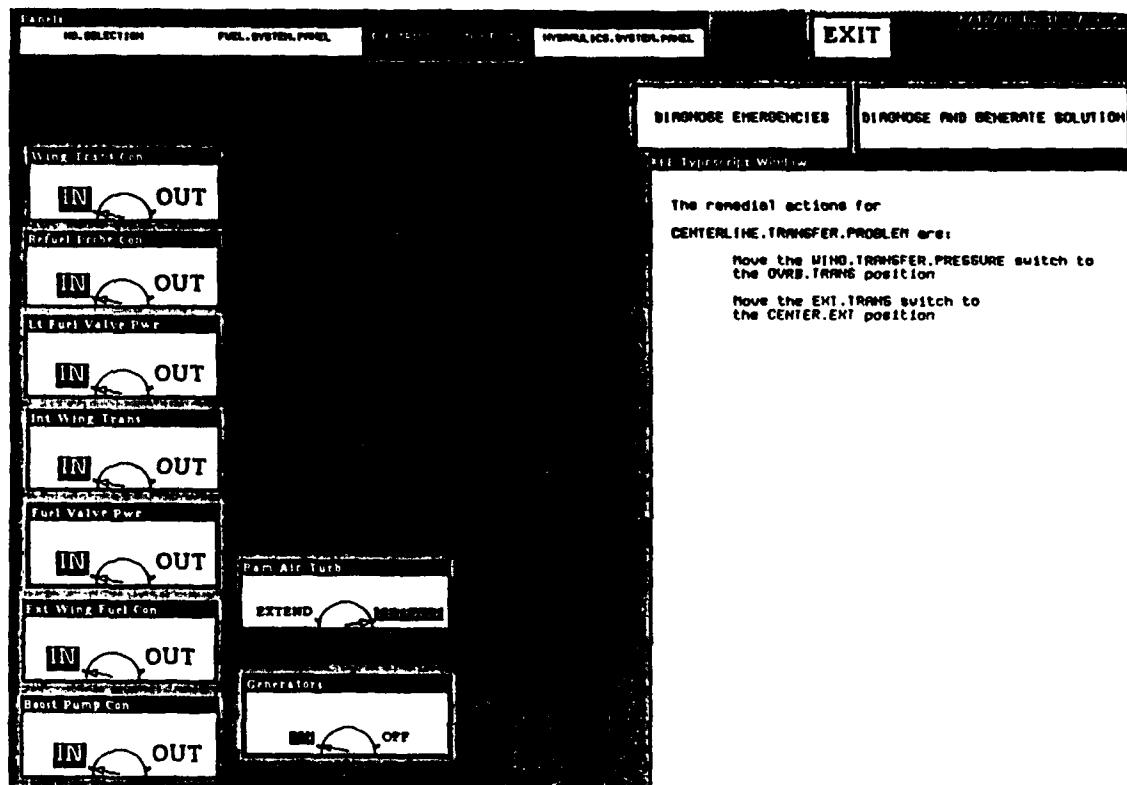
Figure 28 - PEPTR Partial Solution Display
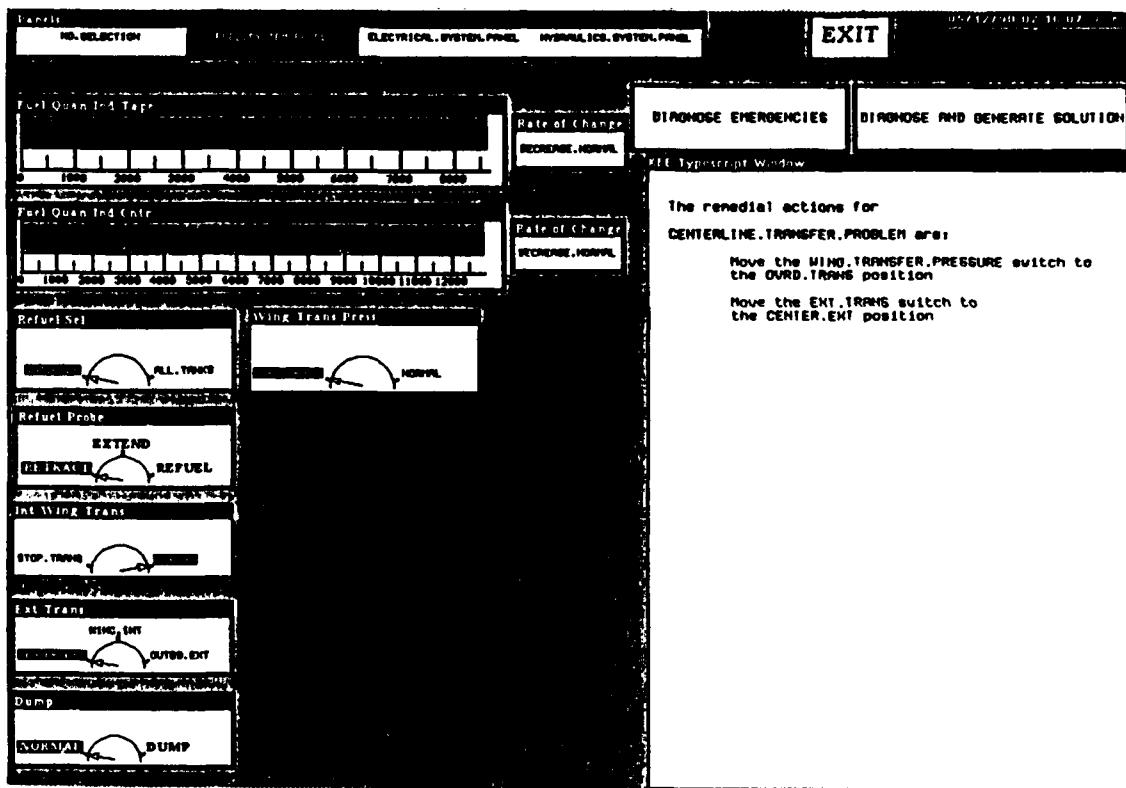
Figure 29 - PEPTR Changing Fuel System State Display

LIST OF REFERENCES

Anderson, J. R., Boyle, C. F., and Reiser, B. J., "Intelligent Tutoring Systems," *Science*, v. 228, 26 April 1985.

Barr, A., and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, v. 1, 2, 3, William Kaufmann, Inc., 1982.

Begg, I. M. and Hogg, I., Authoring Systems for ICAI, in Kearsley, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Co., 1987.

Bourne, J.R., and others, "Intelligent CAI in Engineering: Knowledge Representation Strategies to Facilitate Model-Based Reasoning," *International Journal of Intelligent Systems*, v. 3, 1988.

Bumbaca, F., "Intelligent Computer-Assisted Instruction: A Theoretical Framework," *International Journal of Man-Machine Studies*, v. 29, 1988.

Burton, R. R., Diagnosing Bugs in a Simple Procedural Skill, in Sleeman, D. and Brown, J.S., eds, *Intelligent Tutoring Systems*, Academic Press, Inc., 1982.

Charniak, E., *Artificial Intelligence Programming*, 2d ed., Lawrence Erlbaum Associates, Inc.,1987.

Clancey, W. J., Tutoring Rules for Guiding a Case Method Dialogue, in Sleeman, D. and Brown, J.S., eds, *Intelligent Tutoring Systems*, Academic Press, Inc., 1982.

Dede, C., "A Review and Synthesis of Recent Research in Intelligent Computer-Assisted instruction," *International Journal of Man-Machine Studies*, v. 24, 1986.

Duchastel, P., "ICAI Systems: Issues in Computer Tutoring," *Computers Education*, v. 13, 1989.

Duchastel, P., "Research Directions for ICAI in Canada," *Canadian Artificial Intelligence*, July 1988.

Duchastel, P., Brien, R., and Imbeau, J., "Models of Learning in ICAI," *J. Educational Technology Systems*, v. 17, 1988-89.

Ford, L., "Teaching Strategies and Tactics in Intelligent Computer Aided Instruction," *Artificial Intelligence Review*, v. 1, 1987.

Gevarter, W. B., "The Nature and Evaluation of Commercial Expert System Building Tools," *Computer*, May 1987.

Good, R., "Artificial Intelligence and Science Education," *Journal of Research in Science Teaching*, v. 24, 1987.

Gott, S. P., Bennett, W., and Gillet, A., "Models of Technical Competence for Intelligent Tutoring Systems," *Journal of Computer-Based Instruction*, v. 13, Spring 1986.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. M., STEAMER: An Interactive, Inspectable, Simulation-Based Training System, in Kearsley, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Co., 1987.

IntelliCorp, Inc., *KEE User's Guide*, May 1988.

Jones, M., "Applications of Artificial Intelligence within Education," *Comp. and Maths. with Appls.*, v. 11, 1985.

Kayser, D., "Machine Representation of Knowledge for Computer-Assisted Instruction," *Euromicro Journal*, v. 6, July 1990.

Kearsley, G., Overview, in Kearsley, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Co., 1987.

McCarthy, J., Epistemological Problems of Artificial Intelligence, in Brachman, R. J.. and Levesque, H. J., eds, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., 1985.

Minsky, M., A Framework for Representing Knowledge, in Brachman, R. J.. and Levesque, H. J., eds, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., 1985.

Murray, W. R., "Control for Intelligent Tutoring Systems: A Blackboard-based Dynamic Instructional Planner," *AICOM*, v. 2, June 1989.

Nawrocki, L. H., Artificial Intelligence Applications to Maintc. Training, in Kearsley, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesiey Publishing Co., 1987.

Ohlsson, S., "Some Principles of Intelligent Tutoring," *Instructional Science*, v. 14, 1986.

Park, O., "Functional Characteristics of Intelligent Computer-Assisted Instruction: Intelligent Features," *Educational Technology*, v. 28, June 1988.

Park, O., Perez, R. S., and Seidel, R. J., Intelligent CAI: Old Wine in New Bottles, or a New Vintage?, in Kearsiey, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Co., 1987.

Pliske, D. B. and Psotka, J., "Exploratory Programming Environments for Designing ICAI," *Journal of Computer-Based Instruction*, v.13, Spring 1986.

Quillian, M. R., Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities, in Brachman, R. J. and Levesque, H. J.., eds, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., 1985.

Rowe, N. C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Inc.,1988.

Seidel, R. J., and Park, O., "Instructional Design Principles and AI Techniques for Development of ICAI," *Computers in Human Behavior*, v. 3, 1987.

Seidel, R. J., Park, O.., and Perez, R. S., "Expertise of ICAI: Development Requirements," *Computers in Human Behavior*, v. 4, 1988.

Sleeman, D., "Intelligent Tutoring Systems: A Review," School of Education and Department of Computer Science, Stanford, California, 1984.

Tanimoto, S. L., *The Elements of Artificial Intelligence*, Computer Science Press, Inc., 1987.

Texas Instruments Incorporated, *Explorer Technical Summary*, 1988.

Wenger, E., *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers, Inc., 1987.

Winograd, T., Frame Representations and the Declarative/Procedural Controversy, in Brachman, R. J. and Levesque, H. J.., eds, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., 1985.

Winston, P. H., *Artificial Intelligence*, 2d ed., Addison-Wesley Publishing Company, Inc., 1984.

Woolf, B. P., Intelligent Tutoring Systems: A Survey, in Shrobe, H. E. and the American Association for Artificial Intelligence, eds., *Exploring Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1988.

Woolf, B. P., "Representing Complex Knowledge in an Intelligent Machine Tutor," Computational Intelligence, v. 3, 1987.

Woolf, B. P., Theoretical Frontiers in Building a Machine Tutor, in Kearsley, G., ed, *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Co., 1987.

Woolf, B. P. and McDonald, D. D., "Building a Computer Tutor: Design Issues," *Computer*, September 1984.

Wyer, J., "New Bird on the Branch: Artificial Intelligence and Computer-Assisted Instruction," *Journal of the Association of Education and Training Technology*, v. 21, August 1984.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center   2
    Cameron Station
    Alexandria, VA 22304-6145

2. Library, Code 0142   2
    Naval Postgraduate School
    Monterey, CA 93943-5002

3. Dr. Yuh-jeng Lee   50
    Naval Postgraduate School
    Code 52, Department of Computer Science
    Monterey, CA 93943-5100

4. Dr. Neil C. Rowe   2
    Naval Postgraduate School
    Code 52, Department of Computer Science
    Monterey, CA 93943-5100

5. MAJ George E. Thurmond   2
    P.O. Box 2661
    Parks, LA 70582

6. CPT. Daniel J. Ragsdale   2
    100 Tanglewood Road
    Stratford, CT 06497

7. CPT. John P. Tidd   2
    4126 Sluga Drive
    Newburg, NY 12550

8. HQDA   2
    ATTN: CSDS-AI (Director)
    Pentagon, Room 1D659, WA DC 20310-0200